

**EXTRA**

LA PRIMERA REVISTA DE MSX DE ESPAÑA  
NUMERO ESPECIAL - P.V.P. 275 PTAS (incluido IVA)

***Especial***

# ***Código Máquina***



# SUSCRIBETE HOY MISMO SI QUIERES ESTAR EN VANGUARDIA

La primera revista de MSX de España en tu domicilio cada mes. Por el precio de DIEZ NUMEROS recibirás DOCE. Además tu condición de suscriptor te da derecho a descuentos y ofertas especiales en otros productos. **MANHATTAN TRANSFER, S.A.**

Nombre y apellidos .....	Deseo suscribirme a la revista <b>SUPERJUEGOS EXTRA MSX</b>	<b>Muy importante:</b> para evitar retrasos en la recepción de los números rogamos detalléis exactamente el nuevo número de los distritos postales. Gracias.
Calle ..... N.º .....	a partir del número .....	
Ciudad ..... Tel. ....	<b>FORMA DE PAGO:</b> Mediante <b>talón bancario</b> a nombre de:	<b>TARIFAS:</b>
Provincia .....	<b>MANHATTAN TRANSFER, S.A.</b> C/. Roca i Batlle, 10-12 08023 Barcelona	España por correo normal Ptas. 1.750,- Europa correo normal Ptas. 2.000,- Europa por avión Ptas. 2.800,- América por avión Ptas. 25 USA \$

## NUMEROS ATRASADOS • NUMEROS ATRASADOS



MSX 2.ª Edición  
N.º 1,2,3,4 - 450 PTAS.



MSX5 150 PTAS.



MSX6 150 PTAS.



MSX7-8 300 PTAS.



MSX9 150 PTAS.



MSX10 150 PTAS.



MSX11 150 PTAS.



MSX12-13 300 PTAS.



MSX 14 160 PTAS.



MSX15 175 PTAS.



MSX16 175 PTAS.



MSX 17 175 PTAS.

## ¡LA 1.ª REVISTA DE MSX DE ESPAÑA!

PARA QUE NO TE QUEDES CON LA COLECCION INCOMPLETA SOLO TIENES QUE ENVIAR HOY MISMO EL BOLETIN DE PEDIDO CON TUS DATOS PERSONALES A «SUPER JUEGOS EXTRA MSX» -DPTO. SUSCRIPCIONES C/. Roca i Batlle, 10-12, 08023 Barcelona.

### BOLETIN DE PEDIDO

Deseo recibir los números ..... ds **SUPERJUEGOS EXTRA MSX**  
para lo cual adjunto talón del Banco ..... n.º ..... a la orden de Manhattan Transfer, S.A.

Nombre y apellidos .....

Dirección ..... Tel. ....

Población ..... DP. .... Prov. ....

**«No se admite contrarrembolso»**



# ESPECIAL CODIGO MAQUINA

PVP 275 ptas - Precio sin IVA pts. Canarias 275 ptas.

**MSX EXTRA ES EDITADA POR  
MANHATTAN TRANSFER, S.A.**

**Director Editorial**

Antonio Tello Salvatierra

**Director Ejecutivo**

Birgitta Sandberg

**Coordinador Técnico**

Fco. Javier Guerrero

**Colaboradores Especiales**

Joaquín López

Juan C. González

Fco. Jesús Viceyra

Carlos Rubio

Marcelo T. Helbling

**Diseño y Maquetación**

Félix Llanos

**Redacción, Administración y Publicidad**

Roca i Batlle, 10-12 - 08023 Barcelona

Tel. (93) 211 22 56

**Fotomecánica y Fotocomposición**

Ungraf, S.A.

Pujadas, 77-79 - 08005 Barcelona

**Impresión**

Rotedic, S.A.

Ctra. de Irún Km. 12,450 - 28049 Madrid

**Distribución**

Gestión y Marketing Editorial, S.A.

Eduardo Torroja, 9-11 - Fuenlabrada (Madrid)

Tel. (91) 690 40 01

Todo el material editado es propiedad  
de Manhattan Transfer, S.A.

Prohibida la reproducción total o parcial por  
cualquier medio o soporte sin la debida autorización  
escrita.

## 4 Introducción

*Conceptos básicos. Codificación binaria, Bit. Tamaño de palabra.*

## 7 El ordenador por dentro

*Anatomía del micro. Historia del microprocesador. Defensa de los 8 bits.*

## 10 Lenguajes

*Niveles en los lenguajes de programación. Introducción al lenguaje máquina. Qué es el lenguaje máquina.*

## 12 Memoria

*Ram y Rom. Midiendo la memoria, el Kilobyte. Cómo escribir en la impresora. Poke y Peek.*

## 15 Herramientas

*Numeración hexadecimal.*

## 16 Técnicas

*Las reglas del buen programador. Cómo usar los registros habituales.*

## 18 El ensamblador

*Qué es, para qué sirve y cómo se usa. Descripción resumida de «Gen». Consejos y trucos en la utilización. Conclusiones. Ensamblado. Operaciones que realiza el Z80. Contenido de la memoria.*

## 21 Tablas

*Instrucciones de la CPU Z-80.*

*- clasificadas por mnemónicos*

*- clasificadas por código de operación*

## 27 Programa

*Catálogo para cassettes. Cargador de datos.*

## 30 El Bios

*Rutinas de código máquina. Rutinas del Bios.*

## 36 Tablas de variables

*Variables Rom del sistema. Variables Ram del sistema.*

## 38 Libros

*Código máquina impreso y pret a porter.*

## 40 Programa

*Desensamblador.*



# INTRODUCCION

## CONCEPTOS BASICOS

Si existe una característica común a todos los organismos vivos desde la diminuta célula hasta el mayor de los mamíferos, ésta es la capacidad de comunicarse con otros seres de la misma especie.

Podríamos definir la comunicación de muchas maneras pero siempre haríamos referencia a un intercambio de ideas, de información en suma. Los hombres para intercambiar ideas o información hacemos uso del lenguaje, y cuando necesitamos comunicarnos con las máquinas también hacemos uso de un lenguaje determinado que éstas puedan entender. Elucubrar acerca de estas hipótesis ha sido durante tiempo patrimonio de la filosofía.

Sin embargo todos estos conceptos pasaron de lleno a adquirir el rango de «científicas» gracias a la codificación binaria y a la posibilidad de manipular dicha codificación por medio de sistemas electrónicos que operan a velocidades vertiginosas.

## CODIFICACION BINARIA, BIT

Llamamos codificación binaria (o de dos estados) al sistema que nos permite reducir una serie de conceptos a su mínima expresión. Una serie de dualidades (encendido-apagado, abierto-cerrado, positivo-negativo) que pueden expresarse por símbolos abstractos como 0 y 1. El ordenador reconoce una tensión eléctrica determinada como un 1 y su ausencia como un 0.

Esta dualidad elemental o unidad mínima de información la llamaremos BIT, que no es sino la abreviatura de dígito binario en inglés (Binary digit).

Así pues, un código digital ordinario no es sino un sistema simbólico basado en la mínima expresión de información, el BIT, que compone un lenguaje particular cuya principal característica es la de ser manipulable por un ordenador o hablando con más propiedad, por un circuito digital.

Las máquinas de computación digital poseen sistemas llamados «biestables» —una especie de relé sofisticado— que presentan la peculiaridad de poder tomar dos estados ( $bi=2$ ). Estos dos estados son al-



ternativos, claro está, no pueden estar encendidos y apagados a la vez. Por una propiedad física (electromagnetismo) toman un estado que pueden ser encendido 1 o apagado 0 y mantiene este estado hasta que otra propiedad física (electromagnética) lo altera.

Estas sucesiones de 1 y 0 (encendidos y apagados, cargados y descargados, imantados y no imantados, etc.) son manipulados por la máquina en forma aritmética y convertidos en valores numéricos al sistema de notación binaria.

En código binario sólo existen como dibujo (guarismo) los números 0 y 1. El 2 forma una unidad de orden superior y se dibuja 10, aunque el valor físico absoluto es el mismo. Es decir  $1+1=2$  en base 10 y  $1+1=10$  en base 2. Evidentemente el valor absoluto de 2 en base de 10 es igual a 10 en base de 2. Bajo este principio se generan los siguientes 16 números.



TE 12, (A+16, B+16), 4, 20



y si seguimos veremos que ciertos números decimales necesitan 8 ó 18 bits. p. ej.

Número decimal	Número binario
0	0
1	1
2	10
3	11
4	100
7	111
8	1000 = 4 bits
15	1111 = 4 bits
18	10000
31	11111
32	100000
83	111111
84	1000000
127	1111111
128	10000000 = 8 bits
255	11111111 = 8 bits
258	100000000
511	111111111
512	1000000000 = 10 bits
1023	1111111111 = 10 bits
1024	10000000000
2047	11111111111
2048	100000000000
4095	111111111111
4098	1000000000000
8191	1111111111111
8192	10000000000000
16,383	11111111111111
18,384	100000000000000
32,787	111111111111111
32,768	1000000000000000 = 16 bits
85,535	1111111111111111 = 16 bits

El problema se limita de esta manera a la conversión de cualquier número en base 2 a decimal y viceversa.

**TABLA 3**

En base 10 al número 51.984 representa según el siguiente desglose:

**4 unidades** =  $4 \times (10^0) = 4$

**8 decenas** =  $8 \times (10^1) = 80$

**9 centenas** =  $9 \times (10^2) = 900$

**1 millar** =  $1 \times (10^3) = 1.000$

**5 decenas de millar** =  $5 \times (10^4) = 50.000$

o sea  $4 + 80 + 900 + 1.000 + 50.000 = 51.984$

de modo análogo en base 2, 1.111 será:

**1 unidad binaria** =  $1 \times (2^0) = 1$

**1 decena binaria** =  $1 \times (2^1) = 2$

**1 centena binaria** =  $1 \times (2^2) = 4$

**1 millar binario** =  $1 \times (2^3) = 8$

o sea  $1 + 2 + 4 + 8 = 15$

Número decimal	Número binario
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000



Comprueba esto en la primera tabla adjunta.

Si no es suficiente con este pequeño repaso te rogamos que acudas a un libro de matemáticas.

Como hemos visto en los anteriores ejemplos con 4 bits podemos obtener 2 combinaciones binarias, así pues con 8 bits podemos codificar 256 números decimales diferentes de 0 a 255. Este concepto es importante pues nuestro ordenador MSX trabaja como veremos más adelante con grupos de 8 bits lo cual es una característica común a muchos ordenadores por lo que existe un nombre peculiar para ello "BYTE". Un BYTE es un grupo de 8 BITS contiguos es decir adyacentes, y su importancia radica en que el ordenador siempre manipula grupos de 8 bits y nunca bits sueltos de uno en uno. Por eso cuando definimos el código máquina para el sistema MSX decimos que el tamaño de la palabra de instrucción es de 8 bits.

mérico, sino como un valor ordinal, yendo de 0 a 7. Imaginemos todos estos ordinales como potencia de 2. Por lo tanto el primer bit será 2 elevado a 0 puesto que es la primera posición, es decir 1 ya que cualquier número elevado a 0 nos da 1. Así pues cada bit dentro de un byte tiene un valor determinado según la posición que ocupe.

Ejemplo:

N.º de Bit	7	6	5	4	3	2	1	0
VALOR RELATIVO	128	64	32	16	8	4	2	1
	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$

Recuerde esta numeración para los bits de un byte, pues es estándar utilizado en casi todos los textos de ordenadores.

Tomemos ahora el mayor número construido con 8 bits: 11111111.

Para conocer el valor de este número, sumemos además el 00000000, de modo que con 8 bits podemos representar un total de 256 números. Este es el número de posibles codificaciones que podemos obtener con un BYTE: 256.

Actualmente están apareciendo en el mercado microprocesadores capaces de «entender» códigos escritos de 16 ó 32 bits. El conjunto de bits que la unidad de proceso central puede «entender» y tratar como entidad única, recibe el nombre de PALABRA.

## TAMAÑO DE PALABRA

Las consecuencias que se derivan del número de bits que puede interpretar simultáneamente la CPU son múltiples. Cuanto mayor es el tamaño de la palabra, una CPU aumentará la complejidad de instrucciones que puede decodificar. Ello conlleva unas mejores prestaciones teóricas del microprocesador. Pero como veremos más adelante, cuanto mayor es el número de bits que puede decodificar simultáneamente la CPU, mayor es la cantidad de números de los que puede disponer. El sistema MSX, utiliza el microprocesador (CPU) Z80A con una palabra de 8 bits —o sea un byte—, que está suficientemente probado y esquematizado, de modo que se trata de un microprocesador fiable (pronto nos ocuparemos extensamente del Z80A).

Finalmente piense que en el sistema MSX, un byte corresponde a una palabra por lo que muy a menudo se confunden los términos. Procure que a usted no le ocurra esto.

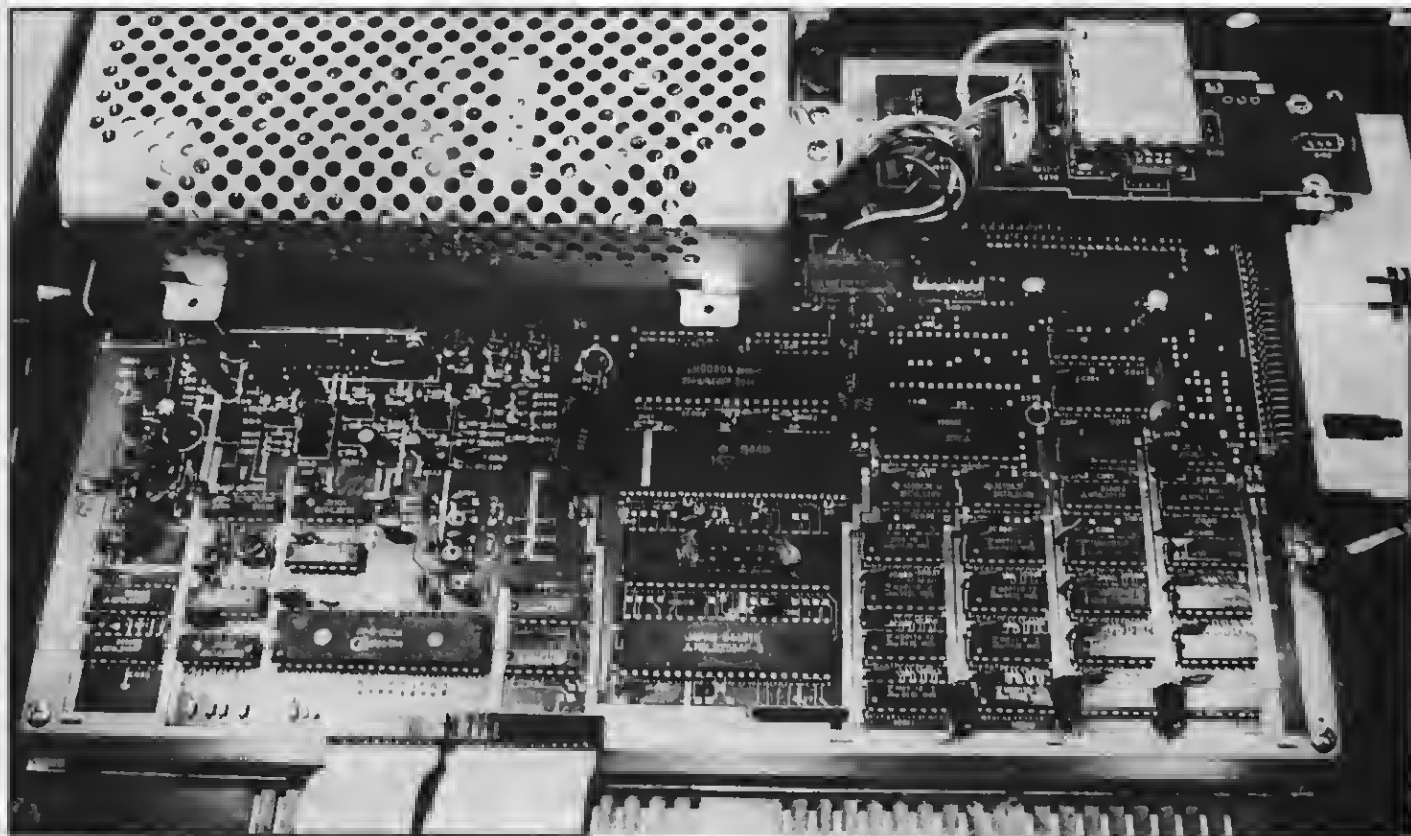
## VALOR RELATIVO DE UN BIT DENTRO DE UN BYTE

Sabemos que un byte está formado por 8 bits. Tomemos entonces estos 8 bits, no como un valor nu-





# ANATOMIA DEL MICRO



Al mirar nuestro ordenador MSX podemos pensar que las principales partes del ordenador son el teclado, la pantalla (televisor o monitor), y el cassette o unidad de disco; nada más lejos de la realidad, pues la fuerza motriz, el corazón y el cerebro de nuestro aparato están en su interior, en sus chips, conglomerado de componentes electrónicos, que posibilitan el funcionamiento correcto de los periféricos antes mencionados.

La palabra «periférico» significa «que está alrededor», así pues teclado, pantalla, cassette, etc. están alrededor, pero no forman parte integral del ordenador.

En realidad podríamos prescindir de la pantalla y comunicarnos con el ordenador sólo a través del teclado como entrada y de la impresora como salida. Asimismo muchos usuarios de la microinformática no disponen de impresora y cuando juegan por medio de los joysticks (mando para juegos) no uti-

lizan para nada el teclado, y sin embargo la función de entrada se realiza igualmente. Todo esto nos demuestra que estos periféricos no son fundamentales para el funcionamiento del ordenador.

La estructura de un ordenador se sustenta sobre la circuitería interna —sus chips— amalgama de componentes electrónicos miniaturizados.

De entre estos componentes el básico es el llamado microprocesador que junto con la memoria conforman ya un ordenador. Es decir, el microprocesador posee en su interior todas las características básicas del ordenador, puede ser programado en código máquina, puede aceptar entradas y controlar salidas, etc. etc... Los demás componentes facilitan y potencian su uso y sus prestaciones.

Sin embargo, en sí mismo el microprocesador o circuito digital programable se encuentra en el corazón de nuestros videojuegos domésticos, estufas, lavadoras automáticas y toda suerte de electrodo-



méticos que poseen la etiqueta de programables. El autor puede comprobar en fecha reciente el funcionamiento de una caña de pescar asistida por un pequeño microprocesador.

Cuando el microprocesador se integra en un sistema más complejo para formar un ordenador recibirá el nombre de CPU (Central Processing Unit): unidad central de proceso.

Sepamos algo más del microprocesador antes de entrar en los detalles técnicos.

## HISTORIA DEL MICROPROCESADOR

En la década de los 50 los ordenadores no tenían el aspecto físico ni la versatilidad que tienen los actuales «pereconales». Podemos decir que inmensos armatostes realizaban las funciones que ahora nos solucionan las pequeñas calculadoras. Además, estos aparatos eran poco fiables, caros de mantener y terriblemente incómodos de manejar. La necesidad de evolucionar estos problemas incentivó la investigación y de esta forma se desarrolló el circuito integrado primeramente, y poco después, en el año 69 de mano de la sociedad INTEL aparece el primer microprocesador que es en suma una CPU completa integrada en un sólo circuito. Este primitivo diseño se perfeccionó y se miniaturizó hasta conseguir en el año 1971 el circuito INTEL 4004. Desde entonces hasta ahora la tecnología ha permitido cada vez miniaturizar más y más los componentes y llegar a los modernos microprocesadores en los que podemos encontrar más de 20.000 transistores sobre un centímetro cuadrado. Se dice que hemos pasado de los circuitos MSI (Medium Scale Integration) que contenían centenas de componentes, a los circuitos LSI (Large Scale Integration) que contienen miles de componentes por centímetro cuadrado y pronto estarán a la orden del día los circuitos VLSI (Very Large Scale Integration) con una integración de más de 100.000 componentes por centímetro cuadrado.

La integración es muy importante para un microprocesador y condiciona su velocidad de funcionamiento, pues cuanto más cerca están los componentes menos tiempo se demora la información en pasar de uno a otro.

El circuito que utiliza el MSX, es el circuito Z80 de Zilog, microprocesador de 8 bits. Decimos pues que las palabras que maneja el Z80 son palabras de un BYTE.

Ya vimos que por «palabra» entendemos el número de bits que puede manejar simultáneamente un ordenador. En nuestro caso se puede confundir fá-

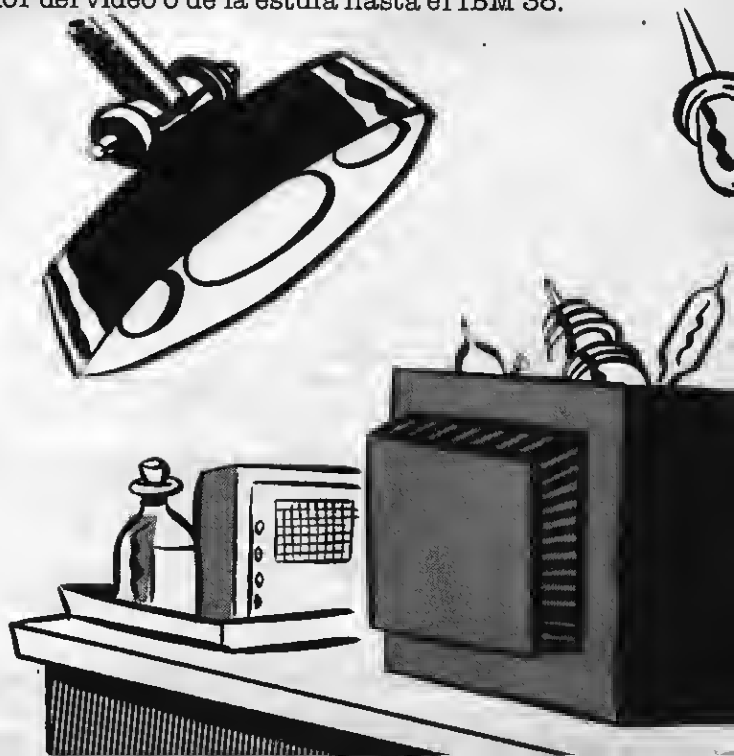
cilmente palabra con byte, pues la palabra del Z80 es de 8 bits. Sin embargo otros ordenadores utilizan palabras de 2 o de 4 bytes. Este circuito Z80 nació en principio de la mano de unos técnicos tráfugos de la casa INTEL que intentaron con este diseño emular y mejorar el funcionamiento del circuito que estaba más en boga en aquella época, el 8080 de INTEL.

La popularidad del Z80 fue en auge, hasta tal punto que en el año 1975 se desarrolló un sistema de explotación por parte de Digital Research que contribuyó a aumentar su popularidad. Actualmente, Microsoft ha salido al paso de críticas afirmando que el Z80 no está en absoluto anticuado y que debido a la inteligente organización de la memoria que se utilizó para crear el standard MSX, éste tardará bastante tiempo en quedar desfasado.

Sin embargo, se ha utilizado una táctica para desvirtuar el sistema MSX, al acusar que utiliza un microprocesador de 8 bits, considerado por muchos como anticuado. Nosotros puntualizamos algo en su defensa.

El microprocesador, también conocido como el C.P.U. (Central Processing Unit) busca la primera instrucción del programa, realiza una serie de operaciones que el ordenador ejecuta a una velocidad vertiginosa, realiza miles en un minuto. La combinación de varias de estas operaciones lógicas y el archivo de datos nos dará una operación matemática, aritmética o algebraica usual. Las referidas operaciones son realizadas por medio de las puertas lógicas.

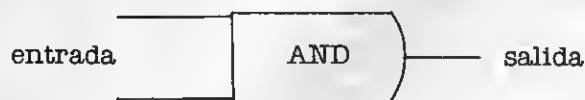
Las puertas lógicas son los elementos básicos de todo dispositivo lógico digital, desde el programador del video o de la estufa hasta el IBM 38.



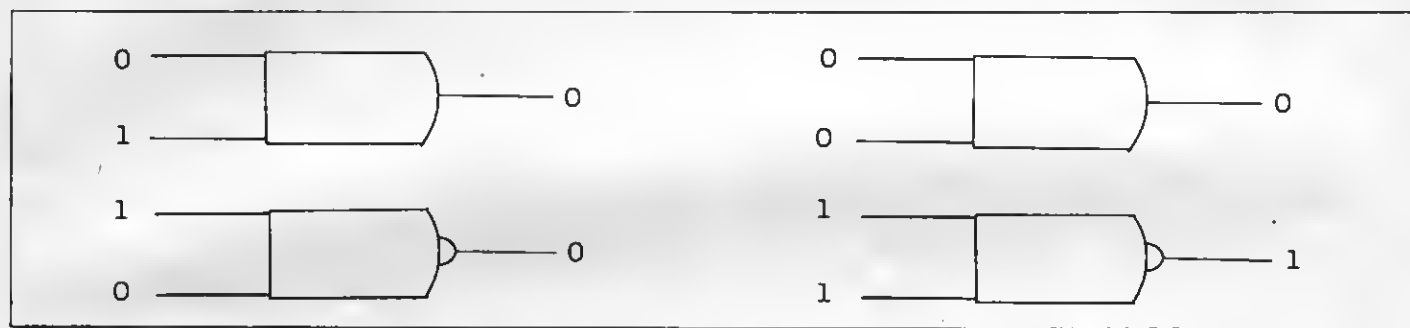




Estos son circuitos que tienen dos, o más entradas y una sola salida digital dependiendo de los datos digitales de la entrada, y cuando decimos digitales, recuerda que hablamos de impulsos eléctricos (1 o de su ausencia 0), es decir, la puerta lógica AND sería así:



si introducimos números digitales, los resultados son los siguientes:



Todo esto se puede reunir en la siguiente tabla de verdad:

AND	0	1
0	0	0
1	0	1

Estas puertas se agrupan formando «chips». Con la suficiente cantidad de puertas AND, OR e inversores podríamos, en teoría, construir cualquier ordenador.

## DEFENSA DE LOS 8 BITS

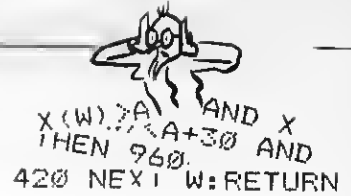
Los ordenadores de 8 bits han llegado a un grado de perfeccionamiento insospechado debido tanto al sofisticado desarrollo de gran cantidad de sus componentes como al alto nivel alcanzado en el diseño de sus circuitos y a la arquitectura de los equipos.

En el caso de los ordenadores de 16 bits todos estos aspectos anteriormente mencionados están aún en vías de desarrollo y en consecuencia muchos aspectos que en el campo de los 8 bits están perfectamente resueltos, en el de los 16 bits aún plantean problemas debido a la falta de componentes adecuados.

Los ordenadores de 8 bits con relación a los de 16 bits conllevan la ventaja de una mayor calidad de software de base (o de sistema), de una superior evolución de los diseños de equipo, resultado de la experiencia acumulada por parte de los fabricantes y comercializadores, asimismo el menor costo y la mejor conveniencia de los equipos de 8 bits, sobre todo en informática de gestión.

Ahora bien, en el campo científico, donde los cálculos son complejos y el volumen de datos es reducido, adquiere primordial importancia la velocidad de cálculo. Pueden resultar más idóneos los procesadores de 16 bits, habida cuenta que el software utilizado en estos casos no es nunca standard, pues sus aplicaciones responden a casos específicos.





# NIVELES EN LOS LENGUAJES DE PROGRAMACION

Los programas que pueden ser directamente ejecutados por un microprocesador están almacenados en lenguaje máquina; no obstante, los programas no se acostumbran a escribir en lenguaje máquina, sino en un lenguaje de más fácil uso para el programador.

Los lenguajes de programación podemos clasificarlos en cuatro clases o niveles:

1.º NIVEL—Lenguajes máquina (Número binario)

2.º NIVEL—Lenguajes simbólicos directos (escritos en mnemónicos, correspondencia uno a uno entre instrucción en mnemónico y número binario. (ASSEMBLER)

3.º NIVEL—Lenguajes de alto nivel funcionales o algoritmos (escritos con mnemónicos, cada instrucción se convierte en un conjunto de instrucciones máquina, FORTRAN, ALGOL, PL/1)

4.º NIVEL—Lenguajes de alto nivel, conversacionales o dialógicos (de funcionalidad parecida a la anterior pero en que son interactivas la ejecución y la creación o modificación de instrucciones (LOGO, BASIC).

Los niveles de programación aumentan con el paso de los años y tienden a ser más complejos, más dialogantes, menos técnicos. Se dice que son idiomas de alto nivel o enfocados al problema, al contrario de los más antiguos, de nivel más bajo y más enfocados u orientados hacia la máquina.

En resumen, a medida que avanza la técnica de la construcción física de las máquinas, se aumenta el desarrollo interior del sistema de símbolos que nos permite utilizar la máquina de una forma apropiada. Una instrucción en BASIC (uno de los idiomas de más alto nivel), una vez traducida nos da una larga lista de instrucciones máquina.

Un idioma de alto nivel está forzosamente soportado por rutinas e instrucciones escritas en un nivel más bajo. Todos los idiomas informáticos están basados en el lenguaje máquina, peculiar del microprocesador que utiliza. Es muy importante que distingamos desde el principio, la diferencia entre el len-

guaje Asembler (también llamado Ensamblador) y el Código Máquina, pues suelen confundirse los términos con frecuencia.

## INTRODUCCION AL LENGUAJE MAQUINA

Los microdomésticos nos son suministrados con un lenguaje que se aproxima mucho más al inglés que al idioma que habla el propio aparato. Programamos los aparatos en BASIC, un lenguaje de ordenador diseñado para hacer la programación general bastante simple. El lenguaje Basic es el medio para llegar a un final y el final es la producción de un código que el ordenador entiende y que le hace reaccionar de la manera que queríamos originalmente. Pero el ordenador no sabe nada de Basic, nada de variables y muy poco de cualquier cosa que pudiéramos considerar útil. Habla un lenguaje, completamente diferente, extremadamente simple, llamado CODIGO MAQUINA.

Cuando programamos un micro MSX en BASIC, sigue necesitando recibir sus instrucciones en su propio código de máquina, que es único para los Z80 e ininteligible para cualquier otro microprocesador.

¿Cómo sabe el ordenador cómo reaccionar a las instrucciones BASIC que le damos?

De la misma manera que intentaríamos entender a otra persona que no hablara nuestro idioma.

Un intérprete no es muy inteligente y es de hecho incapaz de recordar la mayoría de las cosas que ha examinado anteriormente, tanto es así que tiene que hacer exactamente la misma cosa una y otra vez. Esto hace que la interpretación sea muy lenta. Aunque al escribir programas en código máquina ahorramos a nuestro ordenador el tener que utilizar el intérprete (pues hablamos directamente su idioma) y en consecuencia ahorramos trabajo.

## QUE ES EL LENGUAJE MAQUINA

Ya hemos visto como el ordenador únicamente es capaz de manipular señales electrónicas binarias, que representan los estados lógicos 1 y 0, cada instrucción del ordenador está escrita como una serie de 1 y 0 que específicamente caracterizan a esta instrucción y no a otra. A esta representación binaria de las instrucciones de un computador se le llama lenguaje de máquina o código máquina.

MUY PRONTO EN TU QUIOSCO

**MSX 2**

***La primera  
revista de la  
II generación***

**MSX.**

**OTRO PRODUCTO MANHATTAN TRANSFER,<sup>®</sup> S.A.**

# RAM Y ROM

## MEMORIA

Por memoria entendemos cualquier dispositivo que sea capaz de almacenar códigos digitales, bits, (o sea 0 y 1 lógicos). Este almacenamiento nos debe permitir leer y retirar solo el bit o un grupo de ellos.

La tecnología actual determina la creación de varios tipos de memoria. Nuestro ordenador para su funcionamiento dispone de dos de estos tipos.

1.º—**MEMORIA RAM o Random Acces Memory** traducido como memoria de acceso aleatorio y que en español conocemos como memoria de lectura-escritura. En ella podemos almacenar y retirar información codificada digital (series de 1 y 0 lógicos); su peculiaridad es que la información desaparece una vez desconectada la alimentación eléctrica. Por ello es necesario disponer de memorias exteriores o periféricas en las que guardar la información que disponemos en la RAM. Por ej. cassette, diskette o cartucho.

2.º—**MEMORIA ROM o Read Only Memory.** Memoria de sólo lectura que puede ser leída repetidamente, pero su contenido no puede ser modificado (no podemos escribir en ella). Es en este tipo de memoria donde el fabricante guarda la información necesaria para el funcionamiento de nuestro ordenador. Este tipo de memoria no se pierde cuando desconectamos nuestra máquina de la corriente eléctrica.

La memoria de un ordenador está organizada por direcciones. En los ordenadores MSX cada una de las direcciones de memoria contiene 8 instrucciones elementales: 8 BITS (un BYTE).

El chip Z80 puede direccionar, o acceder a 65,536 direcciones de memoria ( $2^{16}$ ). Con un sencillo cálculo podremos apreciar, que son necesarios 16 bits para poder «nombrar» a estas direcciones. Es por ello, que el bus de direcciones dispone de 16 hilos.

Esto plantea un pequeño problema. Al trabajar con palabras de 8 bits y necesitar 16 para direccionar una posición de memoria, el microprocesador Z-80 trata las direcciones de memoria de 16 bits como dos bytes de dirección de memoria, un byte HI de 8 bits y un byte LO de 8 bits. Esto se define como sigue:

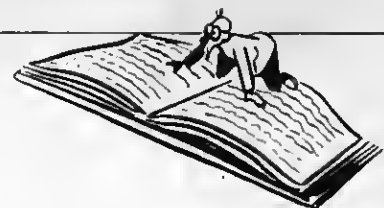
HI byte de dirección alta: los 8 bits más significativos (o los que están más a la izquierda). En forma abreviada H o HI (de HIGH).

LO byte de dirección baja: los 8 bits menos significativos (o los que están más a la derecha). En forma abreviada L o LO (LOW)

Para especificar una posición de memoria se debe especificar los dos bytes de dirección HI y LO, que juntos expresan una dirección de memoria de 16 bits. Este concepto de dirección alta y baja es muy importante, pues constantemente necesitaremos transformar las direcciones de 16 bits en dos palabras de 8 bits. Si queremos decirle al microprocesador que vaya a la dirección 64215 (naturalmente si no decimos nada nos referimos a numeración decimal), tendremos que separar en dos este valor ¿cómo?, pues dividiendo entre 256, pues éste es el peso del bit alto (HI) por su posición, pues como veremos en el gráfico.







formando una palabra de dirección de 16 bytes.

La complicación del proceso no hace sino demostrar una vez más que el sistema decimal es bastante impropio para trabajar con computadoras, sin embargo el sistema binario, se vuelve asimismo impracticable, pues nadie es capaz de recordar series de 1 y 0 agrupadas de 8 en 8. Por ello se desarrollaron dos sistemas de numeración que «funcionan» tan bien como el sistema binario, y que se conocen como Octal y Hexadecimal (Hexa). El BASIC MSX dispone de instrucciones que permiten convertir automáticamente de decimal a binario, Octal y Hexa. De estos dos sistemas, el octal ha sido totalmente desalojado por el hexadecimal, que a partir de aquí denominaremos simplemente Hexa.

Por si tiene interés tanto en la forma con que MSX BASIC reconoce los números binario hexa y octales, así como sobre el sistema octal dispone de sendos apéndices con los N.º 1 y 2.

## MIDIENDO LA MEMORIA EL KILOBYTE

En informática personal, una de las «palabras mágicas» que escuchamos continuamente es el término «K», abreviatura de Kbytes o Kilobytes, lo cual tendría que equivaler a 1000 bytes (kilo quiere decir 1000), sin embargo un Kbyte equivale a 1024 bytes. ¿A qué se debe?...

Como ya hemos dicho el sistema decimal resulta poco operativo para trabajar con el ordenador, por lo que se busca una unidad superior al byte cercana a 1000, pero que fuera redonda en hexa, así se forma el acuerdo de conveniencia de hacer el kilo informático igual a 400 H, o sea 1024 en decimal.

$$400 \text{ H} = 0 \times (16^0) + 0 (16^1) + 4 \times (16^2) = 4 \times 256 = 1024$$

¿De cuántos K dispone nuestro ordenador?

Una de las características no estándar del sistema MSX es la memoria libre para usuario, es decir la RAM, en la cual cada marca puede en este aspecto tomarse sus libertades.

Las especificaciones standard en este sistema son:

**Ram 80K**

**35K Sistema operativo**

**29K Usuario**

**16K Video**

El valor del byte alto, equivale a 256 veces el del byte bajo pues, la unidad del byte alto, su bit N.º 0, ya vale 256 del byte bajo, cuando estos están unidos

**2.º BYTE**

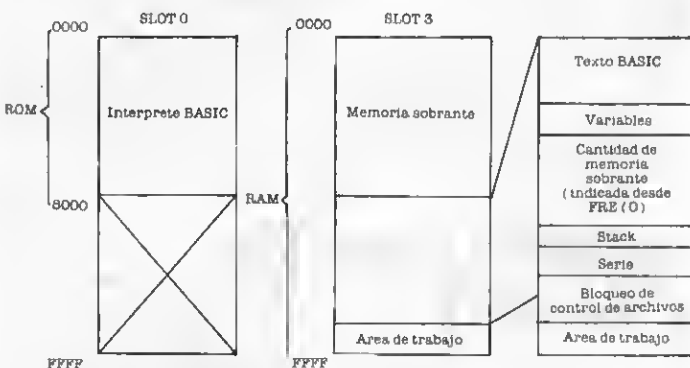
15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0

**1.º BYTE**

7	6	5	4	3	2	1	0
(128)	(64)	(32)	(16)	(8)	(4)	(2)	(1)



gráfico. La porción que va desde **Basic** hasta **serie** puede usarse libremente en Basic. Seguidamente viene el área de carga del MSX, en la cual no se puede cargar programas ni variables. Aquí está el área de memoria que puede ser empleada por el usuario y el valor contenido en FRE (0) es de 28815. Al introducir programas estos son memorizados en el área de texto. Si se utiliza la función PEEK, se observará el contenido de la memoria durante una veintena de bytes a partir de la posición &H 8000, encontrándose que contienen: 0, 9, 128, 10.



## COMO ESCRIBIR EN LA IMPRESORA POKE y PEEK

Poke y Peek son dos instrucciones del Basic que tienen íntima relación con el lenguaje máquina. La sentencia Poke está formada por dos valores separados por una coma (,). El primer valor corresponde a la dirección o posición de memoria que deseamos modificar y tiene que ser un número comprendido entre 0 (cero) y 65.535, eso dependiendo de la memoria de que disponga tu ordenador. Ya sabemos por que...

El segundo valor es el contenido que deseamos introducir en esta posición de la memoria. Hemos de tener presente que esta expresión representa un único byte. Un byte puede tomar 256 ( $2^8$ ) valores diferentes que constituye la unidad de almacenamiento de datos más pequeña accesible al usuario.

En caso de que el valor del contenido sea mayor de 255 tendremos el «error» **illegal function call**. Si introducimos contenidos en posiciones incorrectas nos podemos encontrar con desagradables sorpresas, por ejemplo, que se nos quede bloqueada la máquina y no quede más remedio que pulsar RESET, con lo que se pierde lo introducido hasta el momento.

La función Peek de alguna manera es la inversa de la función Poke. Es decir, Peek nos da el contenido de la dirección de memoria que especifiquemos.

En esencia es una herramienta que permite ver al contenido de una posición deseada.

Es evidente que no podemos alterar los contenidos de la ROM, pero podemos intentarlo, ¿de acuerdo?

El programa que adjuntamos a continuación te será útil para observar zonas de la memoria, mediante la utilización de estas órdenes BASIC.

```

10 SCREEN J:COLOR 1,14:KEY OFF
20 WIDTH 35
30 INPUT"DIRECCION INICIAL (EN DECIMAL)";D
40 IF D<0 OR D>65535! THEN GOSUB 240
50 D$=STRING$(4-LEN(HEX$(D)),"0")+HEX$(D)
60 PRINTD$;"";
70 FOR I = 0 TO 7
80 II=PEEK(D+I):II$=STRING$(2-LEN(HEX$(II)),"0")+HEX$(II)
90 PRINTII$;"";
100 NEXT I
110 FOR I=0 TO 7
120 II=D+I
130 II=PEEK(II)
140 IF II<32 THEN II=46
150 IF II>120 THEN II=46
160 PRINTCHR$(II);
170 NEXT I
180 PRINT
190 D=D+8
200 D$=INKEY$
210 IF D$="" THEN 30
220 GOTO 50
230 END
240 PRINT:PRINTTAB(8)"*";"INTRODUCCION ERRONEA";"*"
250 GOSUB 260:RETURN
260 PRINT:PRINTTAB(10)"PULSA UNA TECLA"
270 Z$=INKEY$:IF Z$="" THEN 270
280 RETURN
  
```



# NUMERACION HEXADECIMAL

Como se ha podido apreciar no es nada fácil descomponer un número decimal, superior a 255 en dos bytes (uno alto y otro bajo). Para solucionar este tipo de inconvenientes, se ha introducido el concepto de numeración hexadecimal, con lo que la conversión en un sentido u otro es automática.

Un número binario emplea sólo dos cifras (dígitos) y un número en notación decimal emplea 10. Para representar los números hexadecimales se utilizan 16 cifras y exactamente:

0, 1, 2, ..., 8, 9, A, B, C, D, E, F

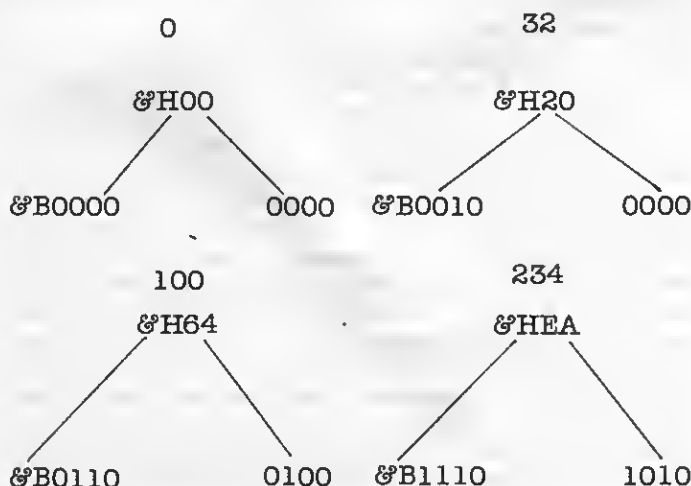
A representa 10, B vale 11 hasta F que vale 15; los valores desde 0 a 15 se representan con un solo carácter o cifra hexadecimal. Los números de dos cifras o sea mayores de 15, son ordenados de la siguiente manera:

10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  
20... 2A 2B 2C 2D 2E 2F 30... 3F 40... 4F  
50 FF 100

El valor decimal de 8F es:

$$8 \times 16^1 + F \times 16^0 = 8 \times 16 + F = 128 + F \equiv 128 + 15 = 143$$

En tanto que 16 es representado por  $2^4$ , cuatro cifras binarias pueden ser representadas por una sola cifra hexadecimal, como puede apreciarse en el siguiente ejemplo de traducción de hexadecimal a binario; los números decimales 0, 32, 100 y 234.



Para facilitar aún más las cosas, los bytes suelen representarse con una pequeña separación entre grupos de cuatro, llamándose a cada grupo nibble, traducible directamente por una cifra Hexa, independientemente de la posición que ocupa dentro del byte, algo que nos facilita con mucho la labor. Por este motivo trabajaremos siempre con este sistema de numeración.

Con la notación decimal el número 65535 no nos salta a la vista ni se nos presenta como importante dentro de la estructura del microprocesador. Sin embargo, si lo traducimos a hexadecimal nos encontramos con

$$\text{\&HFFFF} = 65535$$

que es el tope de memoria que puede direccionar el microprocesador Z80A. La popular K, o kilo informático, se convierte en &H400 o lo que es lo mismo 1024 en decimal. Mediante el programa inserto podemos observar el contenido de las áreas de memoria que deseemos introduciendo una dirección de inicio en hexadecimal.

En este modo, un número binario puede ser convertido fácilmente en número hexadecimal.

Si cuatro bits se pueden representar con un solo dígito Hexa, un número binario de 8 cifras, o sea el valor de un byte, (notación decimal comprendido entre 0 y 255), puede ser representado con sólo dos cifras hexadecimal.

DECIMAL	BINARIO	HEXADECIMAL
0	00000000	00
1	00000001	01
2	00000010	02
3	00000011	03
4	00000100	04
5	00000101	05
6	00000110	06
7	00000111	07
8	00001000	08
9	00001001	09
10	00001010	0A
11	00001011	0B
12	00001100	0C
13	00001101	0D
14	00001110	0E
15	00001111	0F

# LAS REGLAS DE UN BUEN PROGRAMADOR

El buen programador no es el erudito conocedor de todas las instrucciones de CM y su utilidad, naturalmente es fundamental conocer el lenguaje que se utiliza, pero sólo con ello, tanto puede realizarse un programa bueno como uno malo. El resultado dependerá de la TÉCNICA utilizada al programar.

En primer lugar el programa debe estar **ESTRUCTURADO**. Un programa donde el curso del proceso vaya continuamente hacia adelante y hacia atrás hace muy difícil conocer la función de una parte determinada del listado. En un programa desordenado, falta de estructuración, cada modificación y ajuste en vez de solucionar un error introducen otros nuevos.

Una de las reglas fundamentales para estructurar un programa es evitar los saltos. Una vez finalizado el proceso se debe intentar disponer las instrucciones, si no lo están ya, en un orden que evite los saltos, ello no quiere decir que en un programa no existan partes comunes que se reutilicen saltando desde otros puntos, pero, deben reducirse al máximo, puesto que si no el programa pierde claridad.

Relacionado con la característica anterior, el programa debe ser **MODULAR**. Todo aquello que ejecuta una tarea común debe disponerse junto en el listado, es decir, por módulos que realicen una tarea específica y completa.

Para conseguir una buena modularidad, una de las mejores formas es construir el programa a base de subrutinas cada una de las cuales realiza una de las tareas y cuyo orden de ejecución se determina mediante un lazo principal de instrucciones **CALL**. Este sistema tiene la ventaja de que en el momento en que hay que introducir algo nuevo, tan sólo hay que crear la rutina adecuada e introducir un **CALL** a esta rutina en el lugar preciso del lazo principal.

Otro consejo es empezar siempre por lo más difícil. Ello nos permite, por una parte, tomar una idea de lo que nos proponemos o puede llevar a la práctica, y por otra el programa que supedita a las características de lo más complejo. Si no se realiza de esta manera corremos el riesgo de encontrarnos a medio programa y no poder seguir adelante porque hay algo que no se adapta a esa estructura del programa. Al mismo tiempo debemos procurar por aquellos procesos del programa que se vayan a uti-



lizar más veces y que además suelen coincidir con los más complicados; por ejemplo: colocar y desplazar los objetos en la pantalla, las operaciones con los datos, etc..., de esta manera aseguramos que gran parte de lo que vamos necesitando ya lo tenemos construido. Se dice que en CM cada bit debe ayudar a colocar el siguiente.

En CM se dispone de muy pocas «registros-variables» donde realizar el proceso. Es pues imprescindible disponer de una serie de bytes que contengan las variables, los cuales iremos cargando en registros cada vez que haya que consultarlos o modificarlos.

Para que estas variables internas del programa sean operativas y eficaces hay que procurar poner juntas todas aquellas que están relacionadas, por ejemplo: dirección de un objeto en el archivo de la pantalla, su velocidad, su código, el sentido de su





desplazamiento, etc. Podemos optar por poner juntas todas las variables del programa en una misma zona de RAM, o bien ponerlas antes de cada subrutina. Para utilizar la primera técnica tenemos que dimensionar una zona, suficientemente, antes de empezar, con la segunda. El espacio se va reservando a medida que se necesita. Una ventaja de utilizar variables internas es que los parámetros se pueden consultar desde cualquier punto del programa. Además habrá que tener variables que se refieran a los estados internos del programa en una situación concreta, por ejemplo: un indicador para saber si el resultado de una operación ha de ser sumado a un total parcial o no. Este tipo de variables de uso interno posibilitan una mayor versatilidad a las subrutinas, al permitir realizar unas cosas u otras en función del estado de una determinada variable.



Cuando empezamos el planteamiento de un programa es recomendable dividirlo en partes que se puedan tratar de manera independiente, para posteriormente subdividirlos en otras más concretas. Y así sucesivamente hasta que lleguemos a un nivel en que podamos empezar a programar. Este orden es el inverso al que se sigue para construir las subrutinas. Con esto conseguimos mantener siempre controlado el proceso en general, evitando el riesgo de perderse en una parte de una subrutina y manteniendo una visión global del programa.

## COMO USAR LOS REGISTROS HABITUALES

En parte la eficacia de una rutina depende de que los registros empleados sean los adecuados. Por ejemplo, en una rutina que se empleen como pares de registros BL y HC será más eficaz que si se empleara BC y HL, sencillamente porque el Z80 está orientado para dar facilidades a las parejas BC y HL en lugar de BL y HC.

De igual manera, existen ciertos registros más versátiles que otros. Aquellos datos con los que haya que realizar más cosas diferentes intersean cargarlos siempre en los registros más versátiles. El registro A (acumulador) es el que mejor se adapta a estas exigencias. En cuanto a parejas de registros, el par HL se destaca también por ser el que tiene más posibilidades. Esto hace que sea muy útil para emplearlo como puntero, porque es el único que puede proporcionar la dirección de memoria cuyo contenido puede cargarse en cualquier registro. "LD B, (HL)" existe, mientras que "LD B, (DE)", no.

Los registros más versátiles para utilizar como contadores junto a instrucciones automáticas o semiautomáticas como "DJNZ s" o "LDDR" son el registro B utilizado aisladamente y el par BC.

Si necesitamos un registro para alguna tarea con unos datos diferentes a los que contiene, la mejor manera será guardar su contenido en la pila ("stack") y recuperarlo luego, o bien guardar su contenido en aquellos registros que en ese momento no se utilicen. Para esto es muy útil la instrucción "EX DE, HL" que intercambia los contenidos de las dos parejas dando opción a los datos contenidos inicialmente en DE a todas las posibilidades que ofrece el par HL.

Una utilización adecuada de la pila o "stack" junto con la de los registros puede ser muy útil, pero es conveniente no abusar de ella para almacenar datos porque entonces el hecho de que sólo se accese al último dato almacenado se convierte en una dificultad más que en una ventaja.

# QUE ES, PARA QUE SIRVE Y COMO SE USA

Estoy seguro de que muchos de vosotros habréis construido un programa BASIC cuya longitud superaba las mil líneas. Desgraciadamente, tras las largas horas de trabajo invertidas, se acaba comprobando que no es posible continuar, por falta de memoria, o que su ejecución es terriblemente lenta y que todo el esfuerzo ha resultado baldío, puesto que el resultado es poco menos que impresentable.

Esto, que en realidad, es mucho más frecuente de lo que se cree, es rápidamente olvidado, se acomete otro programa y, una vez más, acaba enredado entre las pistas de una casette.

Los programadores prácticos, que de todo hay, buscan soluciones a sus problemas de velocidad y memoria en otros lenguajes. Quiénes así piensan, están llamados a descubrir el ASSEMBLER que es, sin ningún género de dudas, el lenguaje por excelencia y el único capaz de garantizarnos los mejores resultados. Como contrapartida, el ASSEMBLER resulta lento de programar y sobre todo, muy complicado al principio, máxime si se tiene en cuenta que suele hacer falta más de diez instrucciones para emular una sola de BASIC (las funciones requieren bastantes más). Sin embargo, cuando hayáis conseguido cierta soltura o olvidaréis de los lenguajes de alto nivel y «pensaréis» en ASSEMBLER, lo que, a la larga, reporta muchas satisfacciones. Observad que me estoy refiriendo al ASSEMBLER y no al CÓDIGO MÁQUINA. Este último consiste en introducir directamente en la memoria los números correspondientes a los códigos de operación, que después serán interpretados como instrucciones del Z80 cuando se dirija al microprocesador hacia ellos. Esto recibe el nombre de **código objeto**, en otras palabras: el resultado final que se persigue. Por contra, un ensamblador permite entrar palabras clave, llamadas nemónicos o nemotécnicos, que posteriormente serán traducidas a **código objeto**. El conjunto de instrucciones introducidas con ayuda del ensamblador reciben el nombre de **código fuente**. Por cierto, un consejo a los optimistas que crean ser capaces de prescindir del ensamblador y de programar directamente en código máquina: no lo hagáis. Pensad que es virtualmente imposible introducir los códigos sin cometer equivocaciones y resulta, además, extremadamente



lento, con el agravante de que no existen mensajes de error (el ordenador suele quedarse «colgado» o, simplemente, inicializarse).

Nada mejor para tentaros a probar el ASSEMBLER que describir un ensamblador, concretamente el GEN, programado por la firma DEVPAK y con el anagrama de SONY en la carátula. En realidad más que un ensamblador es «el ensamblador», puesto que es el que ofrece mejores prestaciones con mucha diferencia. Este «toolkit» (programa herramienta) no es nuevo, ya que existen versiones de él para todos los microordenadores populares que usan el Z80 como microprocesador, aunque sabe aprovechar al máximo las excelentes posibilidades del sistema MSX, sobre todo en lo referente al editor de pantalla completa.

## DESCRIPCION RESUMIDA DE "GEN"

Llega al usuario con un manual muy completo y totalmente traducido al castellano. Al cargarlo, aparece en la pantalla un glosario de los comandos admitidos, que suelen consistir en una sola letra correspondiente a la inicial de la tarea que efectúan (sea sí, de los respectivos vocablos ingleses). Así, "I" (de insert) funciona de forma similar al modo auto del BASIC y admite dos parámetros, que deben ser la primera línea y el incremento; "L" sirve para listar; "D" para borrar bloques de líneas; "N" para reenumerar; "M" para mover trozos del programa a otra posición; etc. No obstante, el comando más im-



portante es "A" (de assembly) que se usa para traducir código fuente a objeto, poniéndolo en la dirección especificada por el pseudónimo ORG. Naturalmente no existen las instrucciones como GOTO 10. En su lugar hay que definir las direcciones concretas de memoria con **etiquetas** (lo cual ocurre en casi todos los lenguajes, menos en el BASIC). Estas pueden colocarse en cualquier momento, escribiendo una serie de caracteres empezada con una letra y terminada con dos puntos (:). En el supuesto de que olvidemos alguna etiqueta el ensamblador dará una señal de aviso, al final del ensamblado, del tipo "\*\*WARNING NOMBRE absent\*\*", que significa: CUIDADO NOMBRE ausente.

Termino con una mención a otras funciones que efectúan tareas como la de grabar código fuente, cargar desde la cinta, verificar una grabación, listar por impresora, ejecutar programas desde el ensamblador, informar sobre la situación y la longitud del texto, buscar secuencias de caracteres y un largo etcétera.

## CONSEJOS Y TRUCOS EN LA UTILIZACION DEL ENSAMBLADOR

Ahora voy a presumir que ya tienes un ensamblador y quieres sacarle un buen rendimiento. De no ser así, te aconsejo que te hagas con uno rápidamente y te asegures de que sea capaz de ubicarse en la RAM no accesible al BASIC, de generar **macros**, de efectuar **ensamblados** condicionales y ensamblados desde cinta sin cargar todo el texto en la memoria (para programas muy largos). Obviamente GEN cumple todos estos requisitos, por lo que te lo digo recomendado.

Si ya lo posees, te sugiero que hagas una copia rápida del mismo, usando la grabación a 2400 baudios, y que aproveches para reubicarlo en la dirección 128, que es la mínima posible (sólo accesible a máquinas de 84K). Una vez allí, el ensamblador podrá disponer de toda la memoria útil para la entrada del texto, puesto que él mismo se encargará de paginar la ROM. Ten presente que el código fuente de un programa cuyo código objeto ocupe, digamos, 5K puede tener fácilmente una longitud de 30K. Por cierto, el manual, al menos el mío, dice que la longitud de GEM es de 7800 bytes, cuando en realidad es de unos 9800 bytes. Si pasas esto por alto el ordenador se colgará cuando intentes correr la copia.

El hecho de situar el programa en RAM no accesible al BASIC tiene, además, una enorme ventaja: reducir el número de veces en las que te ves obligado a desconectar el ordenador y a perder todo el trabajo. Piensa que sólo necesitas pulsar el botón de reset

para que el ordenador se desbloquee reiniciándose. Luego, podrás relanzar el ensamblador y seguir trabajando con el **fichero de texto** (código fuente), para corregir el error con toda comodidad. Naturalmente si tu ordenador no tiene reset puedes instalárselo con un poco de habilidad, aunque particularmente prefiero introducir dos cables en la ranura del cartucho, lo que tiene el mismo efecto (¡cuidado!). Este pequeño truco basta para solucionar un gran número de errores, aunque si el "gusano" es de los que llenan toda la memoria de valores incorrectos no será suficiente. Para solucionar este último caso, lo mejor es grabar previamente en una cinta la zona de trabajo que GEN tiene en la parte superior de la RAM. Así podrás cargar las rutinas que conmutan los bancos y que sirven para situar correctamente el ensamblador en la memoria, sin importar que éstas hayan sido deterioradas por el error.

Restan por considerar los errores que contaminan la pila. Estos ya son cuestión de suerte, puesto que una serie de POPE desafortunados pueden hacer que se active la RAM no accesible y que el control vuelva a GEN, con fatales consecuencias para el programa y el texto que estaba tratando. En la práctica, estos errores son los menos, por lo que se puede asegurar que es prácticamente "incolable".

En lo tocante a las múltiples opciones de ensamblado el manual es bastante claro. Naturalmente tú has de seleccionar la más útil en cada momento. La opción 36 es la más rápida, por lo que es la que se emplea con más frecuencia. Por contra, si deseas hacer un listado por impresora la apropiada es la 9, que muestra las etiquetas al final del texto. Sin embargo, los programas cuyo código objeto ocupará más de 10K suelen partirse en trozos más pequeños y manejables, a fin de no entretener la programación. En este último caso la opción 45 te listará solamente las etiquetas, que deberás tener en cuenta para que las diferentes partes del programa puedan comunicarse entre ellas.

## CONCLUSIONES

- Un lenguaje casi insustituible: el ASAMBLER.
- Una inversión acertada: un ensamblador.
- Un ensamblador excelente: GEN.
- Una gran comodidad: situar el ensamblador en la RAM paginada.
- Un buen invento: el botón de reset.
- Se me olvidaba: el manual de GEN asegura que es posible, gracias al comando "Y", seleccionar el número de líneas para cada página de los listados que salen por impresora, pero no es cierto.
- El programa perfecto aún está por escribir...



## ENSAMBLADO

Este ensamblado consiste en un listado generado por el ordenador a través de la pantalla o impresora en el que se encuentra el programa fuente (escrito en mnemónicos) con la traducción de cada instrucción al código máquina y con las direcciones de memoria que ocupa cada instrucción a partir de la dirección que se ha tomado como origen al realizar el programa fuente y con las direcciones de las «etiquetas» correctamente situadas en sus lugares respectivos. Esta característica de los ensambladores es fundamental puesto que permite a cada nuevo ensamblaje obtener las direcciones corregidas de todas las sentencias o instrucciones del programa. Con ello se consigue no tener que preocuparse de dichas direcciones y decir sencillamente «ealta a la posición marca» siendo «marca» una etiqueta que define la posición de una determinada sentencia del programa y que si se introducen nuevas instrucciones antes o después y su posición tanto relativa como absoluta cambia, cuando vuelve a ensamblarse queda automáticamente corregido.

Además del listado se obtiene un «programa objeto» en lenguaje máquina, que puede grabarse/ leerse mediante las instrucciones BLOAD, y BSAVE.

## Grupo de Operaciones que realiza el Z80

Entendemos por operación una acción específica que un microprocesador efectuará siempre que lo dicte una instrucción. El número de distintas operaciones que un computador puede efectuar y la velocidad con que puede hacerlo dan una medida de su «potencia»\*. (Ver detrás).

- Operaciones de transferencia de información.
- Operaciones aritméticas.
- Operaciones lógicas.
- Operaciones de subrutinas.
- Operaciones de entrada-salida (E/S)
- Operaciones de incremento-decremento.
- Operaciones de salto.
- Otras operaciones varias.

Recordemos que un byte es un grupo de ocho bits contiguos que ocupan una sola posición de memoria. Muchas instrucciones requieren un solo byte, pero otras exigen dos, tres, o incluso cuatro bytes sucesivos para que puedan ser ejecutadas. Son las llamadas instrucciones multi-byte.

El número de bytes requerido por una instrucción está estrechamente relacionado con la com-

plejidad de la instrucción y con la información que ésta requiera. Las instrucciones de dos y tres bytes tienen bytes que aparecen en posiciones sucesivas de memoria. El primer byte de instrucción se emplea para identificar de qué tipo de instrucción se trata; así sabrá inmediatamente lo que significan los restantes bytes de la instrucción.

## CONTENIDO DE LA MEMORIA

Todo lo que hace el microprocesador con respecto a la memoria, lo hace de ocho en ocho bits, por eso cuando tenemos un programa en código máquina dentro de una zona de la memoria, existen cinco tipos diferentes de información que se pueden almacenar en la memoria:

Códigos de operación de ocho bits.

Bytes de datos de ocho bits.

Códigos de dispositivo de ocho bits.

Bytes de dirección Bajos de ocho bits.

Bytes de dirección Altos de ocho bits.

En un programa en lenguaje máquina, simultáneamente almacenamos códigos de instrucción, bytes de datos, códigos de dispositivo y bytes de dirección, en la misma memoria. El microprocesador los distingue según el orden en que aparece la información. Un programa arranca en una dirección de memoria escogida previamente y después procede, operación por operación, hasta una dirección final de memoria. Los códigos de operación siempre dicen lo que se espera en el programa, es decir, si el próximo byte de memoria es de datos, de dirección, de dispositivo u otro código de operación.

### CÓDIGO DE OPERACION\*\*

El primer byte de una instrucción es siempre un código de operación que indica la acción específica que efectuará el Z80.

Las acciones pueden ser de:

Transferencia de datos.

Operaciones aritméticas.

Operaciones lógicas, operaciones de bifurcación.

Operaciones con el stack.

Operaciones E/S.

Operaciones de control de máquina.

### BYTE DE DATOS

El byte de datos es un número binario de ocho bits que la CPU emplea en una operación aritmética o lógica, o almacena en la memoria. Este dato decodificado puede ser una letra, un dibujo, etc, etc, pero la máquina siempre lo tiene que recibir como series de 8 bits.





# Instrucciones de la CPU Z-80 clasificadas por mnemónico

CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE
00	NOP	39	ADD HL, SP	72	LD (HL), D	AB	XOR E
018405	LD BC, NN	3A8405	LD A, (NN)	73	LD (HL), E	AC	XOR H
02	LD (BC), A	3B	DEC SP	74	LD (HL), H	AD	XOR L
03	INC BC	3C	INC A	75	LD (HL), L	AE	XOR (HL)
04	INC B	3D	DEC A	78	HALT	AF	XOR A
05	DEC B	3E20	LD A, N	77	LD (HL), A	B0	OR B
0820	LD B, N	3F	CCF	78	LD A, B	B1	OR C
07	RLCA	40	LD B, B	79	LD A, C	B2	OR D
08	EX AF, AF'	41	LD B, C	7A	LD A, D	B3	OR E
09	ADD HL, BC	42	LD B, D	7B	LD A, E	B4	OR H
0A	LD A, (BC)	43	LD B, E	7C	LD A, H	B5	OR L
0B	DEC BC	44	LD B, H, NN	7D	LD A, L	B8	OR (HL)
0C	INC C	45	LD B, L	7E	LD A, (HL)	B7	OR A
0D	DEC C	48	LD B, (HL)	7F	LD A, A	B8	CP B
0E20	LD C, N	47	LD B, A	80	ADD A, B	B9	CP C
0F	RRCA	48	LD C, B	81	ADD A, C	BA	CP D
102E	DJNZ DIS	49	LD C, C	82	ADD A, D	BB	CP E
118405	LD DE, NN	4A	LD C, D	83	ADD A, E	BC	CP H
12	LD (DE), A	4B	LD C, E	84	ADD A, H	BD	CPL
13	INC DE	4C	LD C, H	85	ADD A, L	BE	CP (HL)
14	INC D	4D	LD C, L	88	ADD A, (HL)	BF	CP A
15	DEC D	4E	LD C, (HL)	87	ADD A, A	C0	RET NZ
1820	LD D, N	4F	LD C, A	88	ADC A, B	C1	POP BC
17	RLA	50	LD D, B	89	ADC A, C	C28405	JP NZ, NN
182E	JR DIS	51	LD D, C	8A	ADC A, D	C38405	JP NN
19	ADD HL, DE	52	LD D, D	8B	ADC A, E	C48405	CALL NZ, NN
1A	LD A, (DE)	53	LD D, E	8C	ADC A, H	C5	PUSH BC
1B	DEC DE	54	LD D, H	8D	ADC A, L	C820	ADD A, N
1C	INC E	55	LD D, L	8E	ADC A, (HL)	C7	RST 0
1D	DEC E	58	LD D, (HL)	8F	ADC A, A	C8	RET Z
1E20	LD E, N	57	LD D, A	90	SUB B	C9	RET
1F	RRR	58	LD E, B	91	SUB C	CA8405	JP Z, NN
202E	JR NZ, DIS	59	LD E, C	92	SUB D	CC8405	CALL Z, NN
218405	LD HL, NN	5A	LD E, D	93	SUB E	CD8405	CALL NN
228405	LD (NN), HL	5B	LD E, E	94	SUB H	CE20	ADC A, N
23	INC HL	5C	LD E, H	98	SUB L	CF	RST 8
24	INC H	5D	LD E, L	98	SUB (HL)	D0	RET NC
25	DEC H	5E	LD E, (HL)	97	SUB A	D1	POP DE
2820	LD H, N	5F	LD E, A	98	SBC A, B	D28405	JP NC, NN
27	DAA	80	LD H, B	99	SBC A, C	D320	OUT (N), A
282E	JR Z, DIS	81	LD H, C	9A	SBC A, D	D48405	CALL NC, NN
29	ADD HL, HL	82	LD H, D	9B	SBC A, E	D5	PUSH DE
2A8405	LD (HL), (NN)	83	LD H, E	9C	SBC A, H	D820	SUB N
2B	DEC HL	84	LD H, H	9D	SBC A, L	D7	RST 10H
2C	INC L	85	LD H, L	9E	SBC A, (HL)	D8	RET C
2D	DEC L	88	LD H, (HL)	9F	SBC A, A	D9	EXX
2E20	LD L, N	87	LD H, A	A0	AND B	DA8405	JP C, NN
2F	CPL	88	LD L, B	A1	AND C	DB20	IN A, (N)
302E	JR NC, DIS	89	LD L, C	A2	AND D	DC8405	CALL C, N
318405	LD SP, NN	8A	LD L, D	A3	AND E	DE20	SBC A, N
328405	LD (NN), A	8B	LD L, E	A4	AND H	DF	RST 18H
33	INC SP	8C	LD L, H	A5	AND L	E0	RET PO
34	INC (HL)	8D	LD L, L	A6	AND (HL)	E1	POP HL
35	DEC (HL)	8E	LD L, (HL)	A7	AND A	E28405	JP PO, NN
3620	LD (HL), N	8F	LD L, A	A8	XOR B	E3	EX (SP), HL
37	SCF	70	LD (HL), B	A9	XOR C	E48405	CALL PO, NN
382E	JR C, DIS	71	LD (HL), C	AA	XOR D	E5	PUSH HL

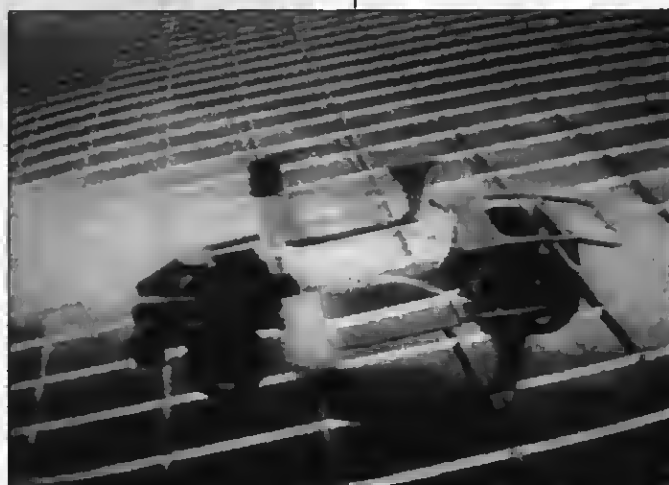


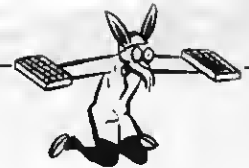
(W)+10 AND 410 IF Y(W)  
Y(W) <E+30 AND +5>E

CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE
E620	AND N	CB26	SRA B	CB70	BIT 6, B	CB80	RES 6, B
E7	RST 20 H	CB29	SRA C	CB71	BIT 6, C	CB81	RES 6, C
E6	RET PE	CB2A	SRA D	CB72	BIT 6, D	CB82	RES 6, D
E9	JP (HL)	CB2B	SRA E	CB73	BIT 6, E	CB83	RES 6, E
EA8405	JE PE NN	CB2C	SRA H	CB74	BIT 8, H	CB84	RES 8, H
EB	EX DE, HL	CB2D	SRA L	CB75	BIT 6, L	CB85	RES 6, L
EC8405	CALL PE, NN	CB2E	SRA (HL)	CB76	BIT 8, (HL)	CB86	RES 6, (HL)
EE20	XOR N	CB2F	SRA A	CB77	BIT 6, A	CB87	RES 6, A
EF	RST 26H	CB36	SRL B	CB78	BIT 7, B	CB88	RES 7, B
FO	RET P	CB39	SRL C	CB79	BIT 7, C	CB89	RES 7, C
F1	POP AF	CB3A	SRL D	CB7A	BIT 7, D	CB8A	RES 7, D
F28405	JPP, NN	CB3B	SRL E	CB7B	BIT 7, E	CB8B	RES 7, E
F3	D1	CB3C	SRL H	CB7C	BIT 7, H	CB8C	RES 7, H
F48405	CALL P, NN	CB3D	SRL L	CB7D	BIT 7, L	CB8D	RES 7, L
F5	PUSH AF	CB3E	SRL (HL)	CB7E	BIT 7, (HL)	CB8E	RES 7, (HL)
F520	OR N	CB3F	SRL A	CB7F	BIT 7, A	CB8F	RES 7, A
F7	RST 30H	CB40	BIT 0, B	CB80	RES 0, B	CB8C	SET 0, B
F6	RET M	CB41	BIT 0, C	CB81	RES 0, C	CB8C	SET 0, C
F9	LD SP, HL	CB42	BIT 0, D	CB82	RES 0, D	CB8C	SET 0, D
FA8405	JP M, NN	CB43	BIT 0, E	CB83	RES 0, E	CB8C	SET 0, E
FB	EI	CB44	BIT 0, H	CB84	RES 0, H	CB8C	SET 0, H
FC8405	CALL M, NN	CB45	BIT 0, L	CB85	RES 0, L	CB8C	SET 0, L
FE20	CP N	CB46	BIT 0, (HL)	CB86	RES 0, (HL)	CB8C	SET 0, (HL)
FF	RST 36H	CB47	BIT 0, A	CB87	RES 0, A	CB8C	SET 0, A
CB00	RLC B	CB48	BIT 1, B	CB88	RES 1, B	CB8C	SET 1, B
CB01	RLC C	CB49	BIT 1, C	CB89	RES 1, C	CB8C	SET 1, C
CB02	RLC D	CB4A	BIT 1, D	CB8A	RES 1, D	CB8C	SET 1, D
CB03	RLC E	CB4B	BIT 1, E	CB8B	RES 1, E	CB8C	SET 1, E
CB04	RLC H	CB4C	BIT 1, H	CB8C	RES 1, H	CB8C	SET 1, H
CB05	RLC L	CB4D	BIT 1, L	CB8D	RES 1, L	CB8C	SET 1, L
CB06	RLC (HL)	CB4E	BIT 1, (HL)	CB8E	RES 1, (HL)	CB8C	SET 1, (HL)
CB07	RLC A	CB4F	BIT 1, A	CB8F	RES 1, A	CB8C	SET 1, A
CB08	RRC B	CB50	BIT 2, B	CB90	RES 2, B	CB8C	SET 2, B
CB09	RRC C	CB51	BIT 2, C	CB91	RES 2, C	CB8C	SET 2, C
CB0A	RRC D	CB52	BIT 2, D	CB92	RES 2, D	CB8C	SET 2, D
CB0B	RRC E	CB53	BIT 2, E	CB93	RES 2, E	CB8C	SET 2, E
CB0C	RRC H	CB54	BIT 2, H	CB94	RES 2, H	CB8C	SET 2, H
CB0D	RRC L	CB55	BIT 2, L	CB95	RES 2, L	CB8C	SET 2, L
CB0E	RRC (HL)	CB56	BIT 2, (HL)	CB96	RES 2, (HL)	CB8C	SET 2, (HL)
CB0F	RRC A	CB57	BIT 2, A	CB97	RES 2, A	CB8C	SET 2, A
CB10	RL B	CB58	BIT 3, B	CB98	RES 3, B	CB8C	SET 3, B
CB11	RL C	CB59	BIT 3, C	CB99	RES 3, C	CB8C	SET 3, C
CB12	RL D	CB5A	BIT 3, D	CB9A	RES 3, D	CB8C	SET 3, D
CB13	RL E	CB5B	BIT 3, E	CB9B	RES 3, E	CB8C	SET 3, E
CB14	RL H	CB5C	BIT 3, H	CB9C	RES 3, H	CB8C	SET 3, H
CB15	RL L	CB5D	BIT 3, L	CB9D	RES 3, L	CB8C	SET 3, L
CB16	RL (HL)	CB5E	BIT 3, (HL)	CB9E	RES 3, (HL)	CB8C	SET 3, (HL)
CB17	RL A	CB5F	BIT 3, A	CB9F	RES 3, A	CB8C	SET 3, A
CB18	RR B	CB60	BIT 4, B	CB9A	RES 4, B	CB8C	SET 4, B
CB19	RR C	CB61	BIT 4, C	CB9A	RES 4, C	CB8C	SET 4, C
CB1A	RR D	CB62	BIT 4, D	CB9A	RES 4, D	CB8C	SET 4, D
CB1B	RR E	CB63	BIT 4, E	CB9A	RES 4, E	CB8C	SET 4, E
CB1C	RR H	CB64	BIT 4, H	CB9A	RES 4, H	CB8C	SET 4, H
CB1D	RR L	CB65	BIT 4, L	CB9A	RES 4, L	CB8C	SET 4, L
CB1E	RR (HL)	CB66	BIT 4, (HL)	CB9A	RES 4, (HL)	CB8C	SET 4, (HL)
CB1F	RR A	CB67	BIT 4, A	CB9A	RES 4, A	CB8C	SET 4, A
CB20	SLA B	CB68	BIT 5, B	CB9A	RES 5, B	CB8C	SET 5, B
CB21	SLA C	CB69	BIT 5, C	CB9A	RES 5, C	CB8C	SET 5, C
CB22	SLA D	CB6A	BIT 5, D	CB9A	RES 5, D	CB8C	SET 5, D
CB23	SLA E	CB6B	BIT 5, E	CB9A	RES 5, E	CB8C	SET 5, E
CB24	SLA H	CB6C	BIT 5, H	CB9A	RES 5, H	CB8C	SET 5, H
CB25	SLA L	CB6D	BIT 5, L	CB9A	RES 5, L	CB8C	SET 5, L
CB26	SLA (HL)	CB6E	BIT 5, (HL)	CB9A	RES 5, (HL)	CB8C	SET 5, (HL)
CB27	SLA A	CB6F	BIT 5, A	CB9A	RES 6, A	CB8C	SET 6, A



CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE
CBF0	SET 8, B	DDCB0566	BIT 2, (IX + d)	EDAB	LDD	FD9E05	SBC A, (IY + d)
CBF1	SET 6, C	DDCB056E	BIT 3, (IX + d)	EDA9	CPD	FDA605	AND (IY + d)
CBF2	SET 6, D	DDCB0666	BIT 4, (IX + d)	EDAA	IND	FDAE05	XOR (IY + d)
CBF3	SET 6, E	DDCB056E	BIT 5, (IX + d)	EDAB	OUTD	FDB605	OR (IY + d)
CBF4	SET 6, H	DDCB0576	BIT 6, (IX + d)	EDBO	LDIR	FDBE05	CP (IY + d)
CBF5	SET 6, L	DDCB057E	BIT 7, (IX + d)	EDB1	CPIR	FDE1	POPIY
CBF6	SET 6, (HL)	DDCB0566	RES 0, (IX + d)	EDB2	INIR	FDE3	EX (SP), IY
CBF7	SET 6, A	DDCB056E	RES 1, (IX + d)	EDB3	OTIR	FDE5	PUSH IY
CBF6	SET 7, B	DDCB0596	RES 2, (IX + d)	EDB6	LDDR	FDE9	JP (IY)
CBF9	SET 7, C	DDCB059E	RES 3, (IX + d)	EDB9	CPDR	FDF9	LD SP, IY
CBFA	SET 7, D	DDCB05A6	RES 4, (IX + d)	EDBA	INDR	FDCB0506	RLC (IY + d)
CBFB	SET 7, E	DDCB06AE	RES 5, (IX + d)	EDBB	OTDR	FDCB050E	RRC (IY + d)
CBFC	SET 7, H	DDCB05B6	RES 6, (IX + d)	FD09	ADD IY, BC	FDCB0516	RL (IY + d)
CBFD	SET 7, L	DDCB05BE	RES 7, (IX + d)	FD19	ADD IY, DE	FDCB051E	RR (IY + d)
CBFE	SET 7, (HL)	DDCB05C6	SET 0, (IX + d)	FD218405	LD IY, NN	FDCB0526	SLA (IY + d)
CBFF	SET 7, A	DDCB05CE	SET 1, (IX + d)	FD228405	LD (NN), IY	FDCB052E	SRA (IY + d)
DD09	ADD IX, BC	DDCB05D6	SET 2, (IX + d)	FD23	INC IY	FDCB053E	SRL (IY + d)
DD19	ADD IX, DE	DDCB05DE	SET 3, (IX + d)	FD29	ADD IY, IY	FDCB0546	BIT 0, (IY + d)
DD218405	LD IX, NN	DDCB05E6	SET 4, (IX + d)	FD2A8405	LD IY, (NN)	FDCB054E	BIT 1, (IY + d)
DD228405	LD (NN), IX	DDCB05EE	SET 5, (IX + d)	FD2B	DEC IY	FDCB0555	BIT 2, (IY + d)
DD23	INC IX	DDCB05F6	SET 6, (IX + d)	FD3405	INC (IY + d)	FDCB055E	BIT 3, (IY + d)
DD29	ADD IX, IX	DDCB05FE	SET 7, (IX + d)	FD3605	DEC (IY + d)	FDCB0566	BIT 4, (IY + d)
DD2A8405	LD IX, (NN)	ED40	IN B, (C)	FD360520	LD (IY + d), N	FDCB056E	BIT 5, (IY + d)
DD26	DEC IX	ED41	OUT (C), B	FD39	ADD IY, SP	FDCB0576	BIT 6, (IY + d)
DD3405	INC (IX + d)	ED42	SBC HL, BC	FD4605	LD B, (IY + d)	FDCB057E	BIT 7, (IY + d)
DD3605	DEC (IX + d)	ED438405	LD (NN), BC	FD4E05	LD C, (IY + d)	FDCB0566	RES 0, (IY + d)
DD360520	LD (IX + d), N	ED44	NEG	FD5605	LD D, (IY + d)	FDCB056E	RES 1, (IY + d)
DD39	ADD IX, SP	ED46	RETN	FD5E05	LD E, (IY + d)	FDCB0596	RES 2, (IY + d)
DD4605	LD B, (IX + d)	ED46	IM 0	FD6605	LD H, (IY + d)	FDCB059E	RES 3, (IY + d)
DD4E05	LD C, (IX + d)	ED47	LD I, A	FD8E05	LD L, (IY + d)	FDCB05A6	RES 4, (IY + d)
DD5605	LD D, (IX + d)	ED46	IN C, (C)	FD7005	LD (IY + d), B	FDCB05AE	RES 5, (IY + d)
DD5E05	LD E, (IX + d)	ED49	OUT (C), C	FD7105	LD (IY + d), C	FDCB05B6	RES 6, (IY + d)
DD6605	LD H, (IX + d)	ED4A	ADC HL, BC	FD7205	LD (IY + d), D	FDCB05BE	RES 7, (IY + d)
DD6E05	LD L, (IX + d)	ED4B8405	LD BC, (NN)	FD7305	LD (IY + d), E	FDCB05C6	SET 0, (IY + d)
DD7005	LD (IX + d), B	ED4D	RETI	FD7405	LD (IY + d), H	FDCB05CE	SET 1, (IY + d)
DD7105	LD (IX + d), C	ED50	IND D, (C)	FD7505	LD (IY + d), L	FDCB05D6	SET 2, (IY + d)
DD7205	LD (IX + d), D	ED51	OUT (C), D	FD7705	LD (IY + d), A	FDCB05DE	SET 3, (IY + d)
DD7305	LD (IX + d), E	ED52	SBC HL, DE	FD7E05	LD A, (IY + d)	FDCB05E6	SET 4, (IY + d)
DD7405	LD (IX + d), H	ED538405	LD (NN), DE	FD8605	ADD A, (IY + d)	FDCB05EE	SET 5, (IY + d)
DD7505	LD (IX + d), L	ED56	IM 1	FD6E05	ADC A, (IY + d)	FDCB05F6	SET 6, (IY + d)
DD7705	LD (IX + d), A	ED57	LD A, I	FD9605	SUB (IY + d)	FDCB05FE	SET 7, (IY + d)
DD7E05	LD A, (IX + d)	ED56	IN E, (C)				
DD6605	ADD A, (IX + d)	ED59	OUT (C), E				
DD6E05	ADC A, (IX + d)	ED5A	ADC HL, DE				
DD9605	SUB (IX + d)	ED5B8405	LD DE, (NN)				
DD9E05	SBC A, (IX + d)	ED5E	IM 2				
DDA605	AND (IX + d)	ED60	IN H, (C)				
DDAE05	XOR (IX + d)	ED61	OUT (C), H				
DDB605	OR (IX + d)	ED62	SBC HL, HL				
DDBE05	CP (IX + d)	ED67	RCD				
DDE1	POPIY	ED66	IN L, (C)				
DDE3	EX (SP), IX	ED69	OUT (C), L				
DDE5	PUSH IY	ED6A	ADC HL, HL				
DDE9	JP (IX)	ED6F	RLD				
DDF9	LD SP, IX	ED72	SBC HL, SP				
DDCB0506	RLC (IX + d)	ED738405	LD (NN), SP				
DDCB050E	RRC (IX + d)	ED76	IN A, (C)				
DDCB0518	RL (IX + d)	ED79	OUT (C), A				
DDCB051E	RR (IX + d)	ED7A	ADC HL, SP				
DDCB0526	SLA (IX + d)	ED7B8405	LD SP, (NN)				
DDCB062E	SRA (IX + d)	EDA0	LDI				
DDCB053E	SRL (IX + d)	EDA1	CPI				
DDCB0846	BIT 0, (IX + d)	EDA2	INI				
DDCB054E	BIT 1, (IX + d)	EDA3	OUTI				





# Instrucciones de la CPU Z-80 clasificadas por código de operación

CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE
BE	ADCA, (HL)	CB44	BIT 0, H	CB71	BIT 8, C	3B	DEC SP
DD8E05	ADCA, (IX + d)	CB45	BIT 0, L	CB72	BIT 8, D	F3	DI
FD8E05	ADCA, (IY + d)	CB4E	BIT 1, (HL)	CB73	BIT 8, E	102E	DJNZ DIS
8F	ADCA, A	DDCB054E	BIT 1, (IX + d)	CB74	BIT 6, H	FB	EI
88	ADCA, B	FDCB054E	BIT 1, (IY + d)	CB75	BIT 8, L	E3	EX (SP), HL
89	ADCA, C	CB4F	BIT 1, A	CB7E	BIT 7, (HL)	DDE3	EX (SP), IX
8A	ADCA, D	BC48	BIT 1, B	DDCB057E	BIT 7, (IX + d)	FDE3	EX (SP), IY
8B	ADCA, E	CB49	BIT 1, C	FDCB057E	BIT 7, (IY + d)	08	EX AF, AF'
8C	ADCA, H	CB4A	BIT 1, D	CB7F	BIT 7, A	EB	EX DE, HL
8D	ADC A, L	CB4B	BIT 1, E	CB78	BIT 7, B	D9	EXX
CE20	ADCA, N	CB4C	BIT 1, H	DB79	BIT 7, C	78	HALT
ED4A	ADCHL, BC	CB4D	BIT 1, L	CB7A	BIT 7, D	ED48	IM 0
ED5A	ADCHL, DE	CB58	BIT 2, (HL)	CB7B	BIT 7, E	ED58	IM 1
ED8A	ADCHL, HL	DDCB0556	BIT 2, (IX + d)	CB7C	BIT 7, H	ED6E	IM 2
ED7A	ADCHL, SP	FDCB0558	BIT 2, (IY + d)	CB7D	BIT 7, L	ED78	INA, (C)
85	ADD A, (HL)	CB57	BIT 2, A	DC8405	CALL C, NN	ED20	INA, (N)
DD8805	ADD A, (IX + d)	CB50	BIT 2, B	FC8405	CALL M, NN	ED40	INB, (C)
FD8805	ADD A, (IY + d)	CB51	BIT 2, C	D48405	CALL NC, NN	ED48	INC, (C)
87	ADD A, A	CB52	BIT 2, D	CD8405	CALL NN	ED50	IND, (C)
80	ADD A, B	CB53	BIT 2, E	C48405	CALL NZ, NN	ED58	INE, (C)
81	ADD A, C	CB54	BIT 2, H	F48405	CALL P, NN	ED60	INH, (C)
82	ADD A, D	CB55	BIT 2, L	EC8405	CALL PE, NN	ED88	INL, (C)
83	ADD A, E	CB5E	BIT 3, (HL)	EC8405	CALL PO, NN	34	INC (HL)
84	ADD A, H	DDCB055E	BIT 3, (IX + d)	CC8405	CALL Z, NN	DD3405	INC (IX + d)
85	ADD A, L	FDCB055F	BIT 3, (IY + d)	3F	CCF	FD3405	INC (IY + d)
C820	ADD A, N	CB5F	BIT 3, A	BE	CP (HL)	3C	INCA
09	ADD HL, BC	CB58	BIT 3, B	DDBE05	CP (IX + d)	04	INCB
19	ADD HL, DE	CB59	BIT 3, C	FDBE05	CP (IY + d)	03	INCB
29	ADD HL, HL	CB5A	BIT 3, D	BF	CPA	0C	INCC
39	ADD HL, SP	CB5B	BIT 3, E	B8	CPB	14	INCD
DD09	ADD IX, BC	CB5C	BIT 3, H	B9	CPC	13	INC DE
DD19	ADD IX, DE	CB5D	BIT 3, L	BA	CPD	1C	INCE
DD29	ADD IX, IX	CB66	BIT 4, (HL)	BB	CPE	24	INCH
DD39	ADD IX, SP	DDCB0586	BIT 4, (IX + d)	BC	CPH	23	INCHL
FD09	ADD IY, BC	FDCB0588	BIT 4, (IY + d)	BD	CPL	DD23	INC IX
FD19	ADD IY, DE	CB67	BIT 4, A	FE20	CPN	FD23	INC IY
FD29	ADD IY, IY	CB60	BIT 4, B	EDA9	CPD	2C	INCL
FD39	ADD IY, SP	CB81	BIT 4, C	ED89	CPDR	33	INC SP
A8	AND (HL)	CB82	BIT 4, D	EDA1	CPI	EDAA	IND
DDA605	AND (IX + d)	CB83	BIT 4, E	EDB1	CPIR	EDBA	INDR
FDA605	AND (IY + d)	CB84	BIT 4, H	2F	CPL	EDA2	INI
A7	AND A	CB85	BIT 4, L	27	DAA	EDB2	INIR
A0	AND B	CB6E	BIT 8, (HL)	35	DEC (HL)	E9	JP (HL)
A1	AND C	DDCB056E	BIT 5, (IX + d)	DD3505	DEC (IX + d)	DDE9	JP (IX)
A2	AND D	FDCB058E	BIT 8, (IY + d)	FD3505	DEC (IY + d)	FDE9	JP (IY)
A3	AND E	CB8F	BIT 5, A	3D	DECA	DA8405	JPC, NN
A4	AND H	CB58	BIT 5, B	05	DEC B	FA8405	JPM, NN
A5	AND L	CB89	BIT 5, C	0B	DEC BC	D28405	JPN, NN
E520	AND N	CB6A	BIT 5, D	0D	DEC C	C38405	JPNZ, NN
CB48	BIT 0, (HL)	CB5B	BIT 5, E	15	DEC D	F28405	JPP, NN
DDCB0546	BIT 0, (IX + d)	CB8C	BIT 8, H	1B	DEC DE	EA8405	JPE, NN
FDCB0548	BIT 0, (IY + d)	CB8D	BIT 5, L	1D	DEC E	E28405	JPO, NN
CB47	BIT 0, A	CB78	BIT 8, (HL)	26	DEC H	CA8405	JP Z, NN
CB40	BIT 0, B	DDCB0576	BIT 8, (IX + d)	2B	DEC HL	382E	JR C, DIS
CB41	BIT 0, C	FDCB0576	BIT 8, (IY + d)	DD2B	DEC IX	182E	JR DIS
CB42	BIT 0, D	CB77	BIT 8, A	FD2B	DEC IY	302E	JR NC, DIS
CB43	BIT 0, E	CB70	BIT 8, B	2D	DECL		





CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE
202E	JR NZ, DIS	4F	LD C, A	EDA8	LDD	CB91	RES 2, C
282E	JR Z, DIS	48	LDC, B	EDB8	LDDR	CB92	RES 2, D
02	LD (BC), A	49	LDC, C	EDA0	LDI	CB93	RES 2, E
12	LD (DE), A	4A	LDC, D	ED80	LDIR	CB94	RES 2, H
77	LD (HL), A	4B	LDC, E	ED44	NEG	CB95	RES 2, L
70	LD (HL), B	4C	LDC, H	00	NOP	CB9E	RES 3, (HL)
71	LD (HL), C	4D	LDC, L	86	OR (HL)	DDCB059E	RES 3, (IX + d)
72	LD (HL), D	0E20	LDC, N	DDB805	OR (IX + d)	FDCB059E	RES 3, (IX + d)
73	LD (HL), E	58	LDD, (HL)	FDB805	OR (IX + d)	CB9F	RES 3, A
74	LD (HL), H	DD5605	LDD, (IX + d)	B7	ORA	CB98	RES 3, B
75	LD (HL), L	FDS805	LDD, (IX + d)	B0	ORB	CB99	RES 3, C
3820	LD (HL), N	57	LDD, A	81	ORC	CB9A	RES 3, D
DD7705	LD (IX + d), A	50	LDD, B	B2	ORD	CB9B	RES 3, E
DD7005	LD (IX + d), B	51	LDD, C	B3	ORE	CB9C	RES 3, H
DD7108	LD (IX + d), C	52	LDD, D	B4	ORH	CB9D	RES 3, L
DD7206	LD (IX + d), D	53	LDD, E	B5	ORL	CBA8	RES 4, (HL)
DD7308	LD (IX + d), E	54	LDD, H	F620	ORN	DDCB05A8	RES 4, (IX + d)
DD7406	LD (IX + d), H	55	LDD, L	EDBB	OTDR	FDCB05A8	RES 4, (IX + d)
DD7505	LD (IX + d), L	1620	LDD, N	EDB3	OTIR	CBA7	RES 4, A
DD380620	LD (IX + d), N	ED5B8405	LD DE, (NN)	ED79	OUT (C), A	CBA0	RES 4, B
FD7705	LD (IX + d), A	118405	LDDE, NN	ED41	OUT (C), B	CBA1	RES 4, C
FD7005	LD (IX + d), B	5E	LDE, (HL)	ED49	OUT (C), C	CBA2	RES 4, D
FD7105	LD (IX + d), C	DD5E05	LDE, (IX + d)	ED51	OUT (C), D	CBA3	RES 4, E
FD7205	LD (IX + d), D	FD5E05	LDE, (IX + d)	ED59	OUT (C), E	CBA4	RES 4, H
FD7305	LD (IX + d), E	5F	LDE, A	ED81	OUT (C), H	CBA5	RES 4, L
FD7406	LD (IX + d), H	58	LDE, B	ED89	OUT (C), L	CBAE	RES 5, (HL)
FD7505	LD (IX + d), L	59	LDE, C	D320	OUT (N), A	DDCB05AE	RES 5, (IX + d)
FD380620	LD (IX + d), N	5A	LDE, D	EDAB	OUTD	FDCB05AE	RES 5, (IX + d)
328405	LD (NN), A	5B	LDE, E	EDA3	OUTI	CBAF	RES 5, A
ED438405	LD (NN), BC	5C	LDE, H	F1	POPAF	CBA8	RES 5, B
ED538405	LD (NN), DE	6D	LDE, L	C1	POPBC	CBA9	RES 5, C
228406	LD (NN), HL	1E20	LDE, N	D1	POPDE	CBA0	RES 5, D
DD228406	LD (NN), IX	88	LDH, (HL)	E1	POPHL	CBAB	RES 5, E
FD228405	LD (NN), IY	DD6805	LDH, (IX + d)	DDE1	POPIX	CBAC	RES 5, H
ED738405	LD (NN), 6P	FD8806	LDH, (IX + d)	FDE1	POPIY	CBAD	RES 6, L
0A	LDA, (BC)	87	LDH, A	F5	PUSHAF	CB88	RES 6, (HL)
1A	LDA, (DE)	80	LDH, B	C5	PUSH BC	DDCB0586	RES 6, (IX + d)
7E	LDA, (HL)	81	LDH, C	D5	PUSH DE	FDCB0586	RES 6, (IX + d)
DD7E05	LDA, (IX + d)	82	LDH, D	E5	PUSH HL	CB87	RES 6, A
FD7E05	LDA, (IX + d)	63	LDH, E	DDE5	PUSH IX	CB80	RES 6, B
3A8405	LDA, (NN)	84	LDH, H	FDE5	PUSH IY	CB81	RES 6, C
7F	LDA, A	65	LDH, L	CB86	RES 0, (HL)	CB82	RES 6, D
78	LDA, B	2620	LDH, N	DDCB0588	RES 0, (IX + d)	CB83	RES 6, E
79	LDA, C	2A8405	LDHL, (NN)	FDCB0588	RES 0, (IX + d)	CB84	RES 6, H
7A	LDA, D	218405	LDHL, NN	CB87	RES 0, A	CB85	RES 6, L
7B	LDA, E	ED47	LDI, A	CB80	RES 0, B	CB8E	RES 7, (HL)
7C	LDA, H	DD2A8405	LDIX, (NN)	CB81	RES 0, C	DDCB08BE	RES 7, (IX + d)
ED67	LDA, I	DD218405	LDIX, NN	CB82	RES 0, D	FDCB05BE	RES 7, (IX + d)
7D	LDA, L	FD2A8405	LDIY, (NN)	CB83	RES 0, E	CB8F	RES 7, A
3E20	LDA, N	FD218405	LDIY, NN	CB84	RES 0, H	CB88	RES 7, B
46	LDB, (HL)	8E	LDL, (HL)	CB85	RES 0, L	CB89	RES 7, C
DD4805	LDB, (IX + d)	DD8E05	LDL, (IX + d)	CB8E	RES 1, (HL)	CB8A	RES 7, D
FD4805	LDB, (IX + d)	FD8E05	LDL, (IX + d)	DDCB058E	RES 1, (IX + d)	CB8B	RES 7, E
47	LDB, A	8F	LDL, A	FDCB058E	RES 1, (IX + d)	CB8C	RES 7, H
40	LDB, B	88	LDL, B	CB8F	RES 1, A	CB8D	RES 7, L
41	LDB, C	89	LDL, C	CB88	RES 1, B	C9	RET
42	LDB, D	8A	LDL, D	CB89	RES 1, C	D8	RET C
43	LDB, E	8B	LDL, E	CB8A	RES 1, D	F8	RET M
44	LDB, H, NN	8C	LDL, H	CB8B	RES 1, E	D0	RET NC
45	LDB, L	8D	LDL, L	CB8C	RES 1, H	C0	RET NZ
0820	LDB, N	2E20	LDL, N	CB8D	RES 1, L	F0	RET P
ED4B8405	LD BC, (NN)	ED7B8405	LD SP, (NN)	CB98	RES 2, (HL)	E8	RET PE
018405	LD BC, NN	F9	LD SP, HL	DDCB0598	RES 2, (IX + d)	E0	RET PO
4E	LDC, (HL)	DDF9	LD SP, IX	FDCB0598	RES 2, (IX + d)	CB	RET Z
DD4E05	LDC, (IX + d)	FDF9	LD SP, IY	CB97	RES 2, A	ED4D	RET I
FD4E05	LDC, (IX + d)	318405	LD SP, NN	CB90	RES 2, B	ED45	RET N



CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE
CB16	RL (HL)	ED42	SBC HL, BC	CBF6	SET 6, (HL)	CB2A	SRAD
DDCB0516	RL (IX + d)	ED52	SBC HL, DE	DDCB05F6	SET 6, (IX + d)	CB2B	SRAE
FDCB0516	RL (IY + d)	ED62	SBC HL, HL	FDCB05F6	SET 6, (IY + d)	CB2C	SRAH
CB17	RLA	ED72	SBC HL, SP	CBF7	SET 6, A	CB2D	SRAL
CB10	RLB	37	SCF	CBF0	SET 6, B	CB3E	SRL (HL)
CB11	RLC	CBC8	SET 0, (HL)	CBF1	SET 6, C	DDCB053E	SRL (IX + d)
CB12	RLD	DDCB05C8	SET 0, (IX + d)	CBF2	SET 6, D	FDCB053E	SRL (IY + d)
CB13	RLE	FDCB05C8	SET 0, (IY + d)	CBF3	SET 6, E	CB3F	SRLA
CB14	RLH	CBC7	SET 0, A	CBF4	SET 6, H	CB36	SRLB
CB15	RLI	CBC0	SET 0, B	CBF5	SET 6, L	CB39	SRLC
17	RLA	CBC1	SET 0, C	CBFE	SET 7, (HL)	CB3A	SRLD
CB06	RLC (HL)	CBC2	SET 0, D	DDCB05FE	SET 7, (IX + d)	CB3B	SRL E
DDCB0506	RLC (IX + d)	CBC3	SET 0, E	FDCB05FE	SET 7, (IY + d)	CB3C	SRLH
FDCB0506	RLC (IY + d)	CBC4	SET 0, H	CBFF	SET 7, A	CB3D	SRL L
CB07	RLCA	CBC5	SET 0, L	CBF8	SET 7, B	96	SUB (HL)
CB00	RLCB	CBCE	SET 1, (HL)	CBF9	SET 7, C	DD9605	SUB (IX + d)
CB01	RLCC	DDCB05CE	SET 1, (IX + d)	CBFA	SET 7, D	FD9605	SUB (IY + d)
CB02	RLCD	FDCB05CE	SET 1, (IY + d)	CBFB	SET 7, E	97	SUBA
CB03	RLCE	CBCF	SET 1, A	CBFC	SET 7, H	90	SUBB
CB04	RLCH	CBC8	SET 1, B	CBFD	SET 7, L	91	SUB C
CB05	RLCL	CBC9	SET 1, C	CB28	SLA (HL)	92	SUBD
07	RLCA	CBCA	SET 1, D	DDCB0526	SLA (IX + d)	93	SUB E
ED6F	RLD	CBCB	SET 1, E	FDCB0526	SLA (IY + d)	94	SUBH
CB1E	RR (HL)	CBCC	SET 1, H	CB27	SLA A	96	SUBL
DDCB051E	RR (IX + d)	CBCE	SET 1, L	CB20	SLA B	D620	SUBN
FDCB051E	RR (IY + d)	CBDE	SET 2, (HL)	CB21	SLA C	AE	XOR (HL)
CB1F	RR A	DDCB05D6	SET 2, (IX + d)	CB22	SLA D	DDAE05	XOR (IX + d)
CB16	RR B	FDCB05D6	SET 2, (IY + d)	CB23	SLA E	FDAE05	XOR (IY + d)
CB19	RR C	CBDE	SET 2, A	CB24	SLA H	AF	XORA
CB1A	RR D	CBDO	SET 2, B	CB25	SLA L	A8	XORB
CB1B	RR E	CBDE	SET 2, C	CB2E	SRA (HL)	A9	XORC
CB1C	RR H	CBDE	SET 2, D	DDCB052E	SRA (IX + d)	AA	XORD
CB1D	RR L	CBDE	SET 2, E	FDCB052E	SRA (IY + d)	AB	XORE
1F	RR A	CBDE	SET 2, H	CB2F	SRA A	AC	XORH
CB0E	RR C (HL)	CBDE	SET 2, L	CB26	SRA B	AD	XORL
DDCB050E	RR C (IX + d)	CBDE	SET 3, B	CB29	SRA C	EE20	XORN
FDCB050E	RR C (IY + d)	CBDE	SET 3, (HL)				
CB0F	RRCA	DDCB05DE	SET 3, (IX + d)				
CB08	RRCB	FDCB05DE	SET 3, (IY + d)				
CB09	RRCC	CBDF	SET 3, A				
CB0A	RRCD	CBDE	SET 3, C				
CB0B	RRCE	CBDA	SET 3, D				
CB0C	RRCH	CBDB	SET 3, E				
CB0D	RRCL	CBDC	SET 3, H				
OF	RRCA	CBDD	SET 3, L				
ED87	RRD	CBE6	SET 4, (HL)				
C7	RSTO	DDCB05E8	SET 4, (IX + d)				
D7	RST10H	FDCB05E8	SET 4, (IY + d)				
DF	RST16H	CBE7	SET 4, A				
E7	RST20H	CBE0	SET 4, B				
EF	RST26H	CBE1	SET 4, C				
F7	RST30H	CBE2	SET 4, D				
FF	RST36H	CBE3	SET 4, E				
C1	RST6	CBE4	SET 4, H				
9E	SBCA, (HL)	CBE5	SET 4, L				
DD9E05	SBCA, (IX + d)	CBE6	SET 5, (HL)				
FD9E05	SBCA, (IY + d)	DDCB05EE	SET 5, (IX + d)				
9F	SBCA, A	FDCB05EE	SET 5, (IY + d)				
96	SBCA, B	CBEF	SET 5, A				
99	SBCA, C	CBE6	SET 5, B				
9A	SBCA, D	CBE9	SET 5, C				
9B	SBCA, E	CBEA	SET 5, D				
9C	SBCA, H	CBE8	SET 5, E				
9D	SBCA, L	CBE9	SET 5, H				
DE20	SBCA, N	CBED	SET 5, L				





# PROGRAMA CATALOGO PARA CASSETTES

Esta rutina sirve para listar todos los programas grabados en una cinta, independientemente de su formato. Se indicará el número de orden, el nombre, el tipo de fichero y la velocidad de transferencia.

Naturalmente el tipo está ligado a la instrucción de carga, que puede ser de tres formas: ASCII (LOAD "CAS."), BASIC (BSAVE), y BYTES (BLOAD).

Creo que encontrarás útil la posibilidad de obtener un catálogo completo de una cinta, para seleccionar los programas contenidos en ella y reagrupar los que consideréis necesarios.

Personalmente la he empleado en algunas de mis cintas y he encontrado grabaciones insospechadas que ya no sabía ni que existían. Como siempre, se incluye un pequeño cargador en DATAS para los que no tengáis ensamblador.

Después de cargar la rutina, deberéis lanzarla con DEFUSR=50005, para salida por pantalla, o con DEFUSR=50000, para salida por impresora.

Hisoft GEN Assembler. Page

1.

Pass 1 errors: 00

C350		10	ORG	50000
CB20		20	BUFFER:	EQU 52000
FEE6		30	RELE:	EQU #FEE6
C350	3E01	40	LD	A, 1
C352	3216F4	50	LD	(<#F416>), A
C355	3E01	60	LD	A, 1
C357	32E7FE	70	LD	(<RELE+1>), A
C35A	2144CB	80	LD	HL, BUFFER+36
C35D	0604	90	LD	B, 4
C35F	CD18C4	100	B0:	CALL PRINT
C362	10FB	110	DJNZ	B0
C364	CD25C4	120	CALL	LF
C367	ED4BAFF3	130	LD	BC, (<#F3AF>)
C36B	05	140	DEC	B
C36C	3E2D	150	LD	A, "--"
C36E	DF	160	B1:	RST #18
C36F	10FD	170	DJNZ	B1
C371	CD25C4	180	CALL	LF
C374	0604	190	INICIO:	LD B, 4
C376	C5	200	B2:	PUSH BC
C377	CDE100	210	CALL	#E1
C37A	382D	220	JR	C, ERROR
C37C	C1	230	POP	BC
C37D	10F7	240	DJNZ	B2
C37F	3AA4FC	250	LD	A, (<#FCA4>)
C382	32E6FE	260	LD	(<RELE>), A
C385	060A	270	LD	B, 10
C387	CD0EC4	280	B3:	CALL LEER
C38A	FED0	290	CP	208
C38C	280C	300	JR	Z, BYTES
C38E	FED3	310	CP	211
C390	280D	320	JR	Z, BASIC
C392	FEEA	330	CP	234
C394	280E	340	JR	Z, ASCII
C396	10EF	350	DJNZ	B3
C398	18DA	360	JR	INICIO
C39A	2120CB	370	BYTES:	LD HL, BUFFER
C39D	180F	380	JR	NOMBRE
C39F	2126CB	390	BASIC:	LD HL, BUFFER+6

C3A2	180A	400		JR	NOMBRE
C3A4	212CCB	410	ASCII:	LD	HL, BUFFER+12
C3A7	1805	420		JR	NOMBRE
C3A9	1E13	430	ERROR:	LD	E, 19
C3AE	C36F40	440		JP	#406F
C3AE	CD0EC4	450	NOMBRE:	CALL	LEER
C3B1	FECF	460		CP	207
C3B3	30F9	470		JR	NC, NOMBRE
C3B5	0606	480		LD	B, 6
C3B7	E5	490		PUSH	HL
C3B8	2138C7	500		LD	HL, 51000
C3BB	77	510	B5:	LD	(HL), A
C3BC	23	520		INC	HL
C3BD	CD0EC4	530		CALL	LEER
C3C0	10F9	540		DJNZ	B5
C3C2	3E20	550		LD	A, " "
C3C4	DF	560		RST	#18
C3C5	3AE7FE	570		LD	A, (RELE+1)
C3C8	3C	580		INC	A
C3C9	32E7FE	590		LD	(RELE+1), A
C3CC	3D	600		DEC	A
C3CD	CD02C4	610		CALL	DIV
C3D0	C630	620		ADD	A, "0"
C3D2	DF	630		RST	#18
C3D3	78	640		LD	A, B
C3D4	CD02C4	650		CALL	DIV
C3D7	78	660		LD	A, B
C3D8	C630	670		ADD	A, "0"
C3DA	DF	680		RST	#18
C3DB	0604	690		LD	B, 4
C3DD	3E20	700		LD	A, " "
C3DF	DF	710	B7:	RST	#18
C3E0	10FD	720		DJNZ	B7
C3E2	2138C7	730		LD	HL, 51000
C3E5	CD18C4	740		CALL	PRINT
C3E8	E1	750		POP	HL
C3E9	CD18C4	760		CALL	PRINT
C3EC	3AE6FE	770		LD	A, (RELE)
C3EF	2138CB	780		LD	HL, BUFFER+24
C3F2	FE30	790		CP	48
C3F4	2803	800		JR	Z, LENTO
C3F6	213ECB	810		LD	HL, BUFFER+30
C3F9	CD18C4	820	LENTO:	CALL	PRINT
C3FC	CD25C4	830		CALL	LF
C3FF	C374C3	840		JP	INICIO
C402	060A	850	DIV:	LD	B, 10
C404	0EFF	860	B6:	LD	C, 255
C406	0C	870		INC	C
C407	90	880		SUB	B
C408	30FA	890		JR	NC, B6
C40A	80	900		ADD	A, B
C40B	47	910		LD	B, A
C40C	79	920		LD	A, C
C40D	C9	930		RET	
C40E	C5	940	LEER:	PUSH	BC
C40F	E5	950		PUSH	HL
C410	CDE400	960		CALL	#E4
C413	3894	970		JR	C, ERROR
C415	E1	980		POP	HL



C416	C1	990	POP	BC
C417	C9	1000	RET	
C418	C5	1010	PRINT:	PUSH BC
C419	0606	1020	LD	B,6
C41B	7E	1030	B4:	LD A,(HL)
C41C	DF	1040	RST	#18
C41D	23	1050	INC	HL
C41E	10FB	1060	DJNZ	B4
C420	3E20	1070	LD	A,32
C422	DF	1080	RST	#18
C423	C1	1090	POP	BC
C424	C9	1100	RET	
C425	3E0D	1110	LF:	LD A,13
C427	DF	1120	RST	#18
C428	3E0A	1130	LD	A,10
C42A	DF	1140	RST	#18
C42B	C9	1150	RET	
CB20		1160	ORG	BUFFER
CB20	20425954	1170	DEFM	" BYTES"
CB26	20424153	1180	DEFM	" BASIC"
CB2C	20415343	1190	DEFM	" ASCII"
CB32	20424155	1200	DEFM	" BAUD."
CB38	20313230	1210	DEFM	" 1200 "
CB3E	20323430	1220	DEFM	" 2400 "
CB44	204EA720	1230	DEFM	" N' "
CB4A	4E4F4D42	1240	DEFM	"NOMBRE"
CB50	20544950	1250	DEFM	" TIPO "
CB56	20424155	1260	DEFM	" BAUD."



## CARGADOR DE DATOS

```

10 FORX=50000!TO50219!
20 READV$:POKEX,VAL("&H"+V$)
30 S=S+PEEK(X)
40 NEXT
50 IFS<>26388THENCLS:BEEP:PRINT"HA
Y UN ERROR"
60 DATA3E,01,32,16,F4,3E,01,32,E7,
FE,21,44,CB,06,04,CD,18,C4,10,FB,C
D,25,C4,ED,4B,AF,F3,05,3E,2D,DF,10
,FD,CD,25,C4,06,04,C5,CD,E1,00,38,
2D,C1,10,F7,3A,A4,FC,32,E6,FE,06,0
A,CD,0E,C4,FE,D0,28,0C,FE,D3,28,0D
,FE,EA,28,0E,10,EF,18,DA,21,20,CB,
18,0F,21,26,CB,18

```

```

70 DATA0A,21,2C,CB,18,05,1E,13,C3,
6F,40,CD,0E,C4,FE,CF,30,F9,06,06,E
5,21,38,C7,77,23,CD,0E,C4,10,F9,3E
,20,DF,3A,E7,FE,3C,32,E7,FE,3D,CD,
02,C4,C6,30,DF,78,CD,02,C4,78,C6,3
0,DF,06,04,3E,20,DF,10,FD,21,38,C7
,CD,18,C4,E1,CD,18,C4,3A,E6,FE,21,
38,CB,FE,30,28,03
80 DATA21,3E,CB,CD,18,C4,CD,25,C4,
C3,74,C3,06,0A,0E,FF,0C,90,30,FA,8
0,47,79,C9,C5,E5,CD,E4,00,38,94,E1
,C1,C9,C5,06,06,7E,DF,23,10,FB,3E,
20,DF,C1,C9,3E,0D,DF,3E,0A,DF,C9

```



# RUTINAS DE CODIGO MAQUINA



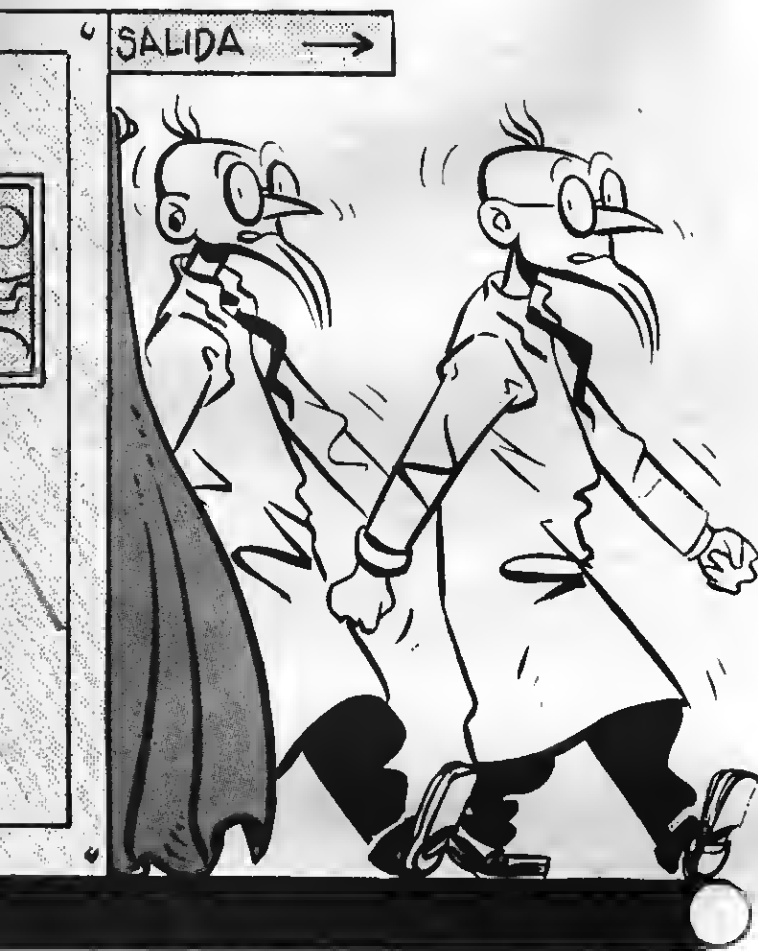
El BIOS (Basic Input Output System) se encuentra en la ROM de tu ordenador. Consiste en una serie de rutinas, escritas en código máquina, capaces de gestionar cosas tan dispares como el teclado, la pantalla, el interfaz de la impresora y el del casete, los puertos de joystick y las ranuras de los cartuchos.

Cualquiera que desmonte un desensamblador habrá comprobado que las posiciones de memoria más bajas de la ROM contienen una serie de saltos absolutos hacia diferentes direcciones (JP dirección). Quizá os hayáis preguntado por qué se desperdicia así tal cantidad de memoria (tres bytes

para cada rutina), ya que sería igual referirse a la posición final, en lugar de pasar por un salto absoluto. Puede bien, esto es en orden a asegurar totalmente la compatibilidad de los diferentes ordenadores MSX, así como de sus futuras mejoras y versiones. Microsoft, la firma creadora del estándar, dictó unas normas a seguir por todos los programadores, que deben ser estrictamente repetidas para que cualquier diferencia en el hardware no repercuta en el funcionamiento del software. Unos ejemplos aclararán mejor este punto. Supón que quieres escribir un dato en el casete, poner en marcha el motor, encender el diodo de las mayúsculas o, simplemente, sacar un carácter por la pantalla. En



cualquiera de estos casos hay una forma directa de obtener el resultado de entrada/salida. No obstante, el mínimo cambio en la asignación comportaría que el ordenador mostrara unos resultados completamente inesperados.



Todo lo anterior conduce a la necesidad de **acceder a las rutinas del BIOS** en lugar de improvisar soluciones de compromiso. Dicho esto, se aprecia claramente la importancia de contar con un mapa de la ROM que dé información de la ubicación y contenido de las rutinas fundamentales. A continuación se detallan, añadiendo, en las más interesantes, una relación de los parámetros de entrada necesarios en cada caso, así como de las modificaciones que efectúan en los registros y en las posiciones de memoria.

Sin duda encontraréis inestimable la ayuda que os brindan las rutinas del BIOS. Desde aquí, el deseo de una fructífera programación.

## LAS RUTINAS DEL BIOS

### Posición: &H0

Esta rutina no necesita parámetros de entrada ni tampoco ofrece ninguno a la salida. Puede ser ejecutada utilizando un restart (RST 0). Su función es la de inicializar el ordenador. Por consiguiente, se llama cuando se quiere empezar de nuevo, cuando se pulsa el botón de reset o, automáticamente, al encender el aparato.

### Posición: &H8 y &H10

Estas rutinas son utilizadas por el intérprete BASIC para analizar los errores de sintaxis, tomar el siguiente carácter o token del programa, etc. Son de poca utilidad, aunque una posible aplicación sería la de construir un BASIC extendido.

### Posición: &HC

Se usa para leer una dirección de memoria de un cartucho determinado. El número de cartucho ha de colocarse en el acumulador y la dirección en el registro HL. Altera AF, BC y DE.

### Posición: &H14

Igual que la anterior pero para escribir.

### Posición: &H18

Es, sin duda, una rutina muy útil. Puede ser llamada con RST 18. Se encargará de sacar el carácter contenido en el acumulador al periférico seleccionado. Si la posición de memoria &HF416 contiene un cero, la salida será a la pantalla. Si &HF416 es distinto de cero, la salida será por impresora. Por último, tienes la posibilidad de escribir en un fichero de disco, cargando &HF864 con la dirección de memoria de dicho fichero, que señalará el dato a mandar. RST 18 no modifica ningún registro. Por otra parte, realiza una llamada al gancho situado en &HFEE4 después de guardar el par AF en la pila. Como puedes intuir, poner un parche en esa dirección te dará la oportunidad de controlar los distintos periféricos a tu antojo.

### Posición: &H1C

Esta rutina se emplea para ejecutar una subrutina de un cartucho.

### Posición: &H20

Puede comparar los registros DE y HL llamando a esta rutina. Aquí tienes el listado:

LD	A, D
CP	H
RET	NZ
LD	A, E
CP	L
RET	

### Posición: &H24

Esta rutina selecciona una página de un cartucho.



### Posición &H28

Es empleada por el intérprete BASIC para conocer el tipo de variable que se está utilizando. Alternativamente se puede leer la dirección &HF663, puesto que siempre se almacena aquí el número de bytes de la variable usada; es decir, dos para las variables numéricas enteras, cuatro para las de precisión sencilla, ocho para las de doble precisión y tres para las cadenas alfanuméricas. Sin embargo, no es seguro que esta dirección se repeta en futuras versiones. Por tanto observa si el flag C está a 0 (tipo 8), el flag M está a 1 (tipo 2), el flag Z está a 1 (tipo 3) o el flag P se encuentra a 0 (tipo 4).

### Posición &H30

Ejecuta una rutina contenida en un cartucho. El byte siguiente al RST 30 debe contener el identificador del cartucho y después debe colocarse la dirección de llamada.

### Posición &H38

Esta rutina es ejecutada 50 veces por segundo, salvo que las interrupciones estén desactivadas. Lo primero que hace es guardar los registros en la pila (incluidos los alternativos y los de índices), por lo que podrá emplearlos todos libremente y sin restricciones. Si pones un parche en la dirección &HFD9A forzarás al sistema operativo a ejecutar una de tus rutinas siempre que se produzca una interrupción. Como puedes ver, esto te da un poder inmenso sobre el ordenador. No modifica ningún registro, pero altera muchas posiciones de memoria, ya que actualiza, entre otras, la variable TIME y las escalas musicales. Asimismo, comprueba las colisiones de los SPRITES, el teclado, etc.

### Posición: &H41

Llamándola haces que la pantalla se desconecte. No obstante, todo lo que escribas se conservará y podrás visualizarlo con la siguiente rutina. Suele ser útil cuando se hace un dibujo muy complicado que se quiere mostrar en pantalla instantáneamente. Modifica los pares AF y BC.

### Posición: &H44

Esta rutina activa la pantalla, por lo que complementa a la anterior. Al igual que aquella, modifica los registros AF y BC.

### Posición: &H47

Se llama a esta rutina para escribir en uno de los registros de estado del procesador de vídeo (VDP). En C debe ponerse el número de registro a escribir y en B el dato en cuestión. Su equivalente en BASIC sería: VDP(C)=B. Es importante emplear esta rutina, en lugar de acceder al VDP directamente, puesto que se encarga de guardar una copia del registro de estado en la RAM del sistema, desde la posición &HF3DF hasta la &HF3E6. Ten presente que estos registros sólo son de escritura y no podrías comprobar los datos una vez mandados. Modifica los

pares AF y BC.

### Posición &H4A

Funciona igual que la instrucción VPEEK del BASIC. Debes cargar la dirección de la RAM de vídeo en el par HL y obtendrás a la salida el resultado en el acumulador. Modifica sólo AF.

### Posición: &H4D

Es idéntica a la anterior sólo que ésta actúa como VPoke. El dato a escribir ha de ponerse en el acumulador.

### Posición: &H50

Dispone el VDP para una operación de lectura. Se mejor pasarla por alto y llamar directamente a la rutina situada en &H59.

### Posición: &H53

Prepara el VDP para una operación de escritura. Al igual que la anterior se mejor olvidarla y acceder a la rutina colocada en &H5C.

### Posición: &H56

Esta rutina llena la RAM de vídeo de un mismo valor contenido en el acumulador. La posición de origen debe encontrarse en HL y la longitud del bloque en BC. Modifica los pares AF y BC. La utilidad de esta rutina es colorear la pantalla rápidamente. Las instrucciones CLS, COLOR, LINE y PAINT la emplean.

### Posición: &H59

Esta rutina traslada un bloque de la RAM del VDP hacia la memoria central. La longitud del referido bloque ha de encontrarse en BC, el destino en DE y el origen en HL. Modifica AF, BC y DE. Tarde o temprano todos los programadores han de encontrarse con esta rutina, por lo que su uso es prácticamente imprescindible.

### Posición: &H5C

La rutina situada en esta dirección tiene un comportamiento análogo a la anterior, con la diferencia de que traslada un bloque desde la memoria central a la RAM de vídeo.

### Posición: &H5F

Esta llamada pone al VDP en uno de los cuatro modos de pantalla. El acumulador deberá contener el modo seleccionado. Su equivalente en BASIC sería SCREEN A. No inicializa los SPRITES. Modifica todos los registros así como las posiciones de memoria &HF3B0, &HF922, &HF924, &HFC4F y &HFCB0.

### Posición: &H62

Esta rutina cambia el color de la pantalla, tomando como nuevos valores las posiciones de memoria siguientes: &HF3E9 (color de la tinta), &HF3EA (color del papel) y &HF3EB (color del borde). Modifica los pares AF, BC y HL.

### Posición: &H69

Su cometido es inicializar todos los SPRITES. Altera todos los registros.



13, (A+8, B+32), 8, 21

#### **Posición: &H6C**

Esta rutina actúa como la instrucción BASIC SCREEN 0. Modifica todos los registros así como las posiciones de memoria que van desde la &HF3DF a la &HF3E5.

#### **Posición: &H6F**

Funciona igual que la anterior pero para el SCREEN 1.

#### **Posición: &H72**

Igual que las anteriores pero para SCREEN 2.

#### **Posición: &H75**

Igual para SCREEN 3.

#### **Posición: &H78**

Inicializa al VDP para trabajar en SCREEN 0, pero sin tocar la RAM de vídeo. Modifica los mismos registros y posiciones de memoria que la rutina situada en &H6C.

#### **Posición: &H7B**

Trabaja igual que la anterior pero para SCREEN 1.

#### **Posición: &H7E**

Igual que las anteriores pero para SCREEN 2.

#### **Posición: &H81**

Lo mismo para SCREEN 3.

#### **Posición: &H87**

Con esta rutina sólo tendrás que cargar un número de SPRITE en el acumulador para que te devuelva la dirección de la VRAM en la que se encuentran los atributos del SPRITE seleccionado, gracias al registro HL. Modifica los pares HL y DE así como los flags.

#### **Posición: &H8A**

Esta rutina te informará del tipo de SPRITE que estás empleando, o mejor dicho: el número de bytes que emplea cada uno de éstos, que pueden ser 8 ó 32. Por tanto, a la salida tendrás en el acumulador una de estas dos cantidades. Además el carry se pondrá a 1 si los SPRITES son del tipo ampliado. Únicamente modifica el par AF.

#### **Posición: &H8D**

Esta rutina escribe el carácter contenido en el acumulador en la dirección especificada por el cursor gráfico (la coordenada X está en &HFCB3 y la Y en &HFCB4), siempre y cuando esté trabajando en SCREEN 2. Sólo modifica las posiciones de memoria &HF92A, &HF923 y &HF92C.

#### **Posición: &H90**

Esta rutina inicializa el Generador Programable de Sonido. No modifica ningún registro, pero altera toda el área de la cola del sonido, que empieza en &HF975 y termina en &HFA74.

#### **Posición: &H93**

Con ella puedes escribir en uno de los registros del PSG. El número de registro ha de colocarse en el acumulador y en E el dato a mandar (comprendido entre 0 y 13). Su equivalente en BASIC sería: SOUND A, E. Esta llamada no modifica ningún registro.

#### **Posición: &H96**

Esta rutina sirve para leer un registro del PSG. El acumulador debe contener el número de registro (comprendido en 0 y 13). Sólo altera el contenido de A.

#### **Posición: &H99**

Se llama a esta rutina para ejecutar la escala musical (caso de haberla). Si en el buffer de sonido no hay ninguna escala escrita el acumulador se cargará con un cero. Modifica los pares AF y HL, así como las posiciones de memoria &HFB3F y &HFB40.

#### **Posición: &H9C**

Comprueba si las teclas de función están activas en la pantalla. En caso afirmativo, analiza las teclas SHIFT, para mostrar el contenido de las funciones F6 y F10, si están puleadas. Esta rutina pondrá el flag Z a 1 si no hay ninguna tecla apretada. Únicamente modifica AF.

#### **Posición: &H9F**

Esta rutina es de gran importancia. Su cometido es coger un carácter del buffer del teclado. Si este buffer está vacío enseñará el cursor y esperará hasta que se pulse una tecla. A la salida, el acumulador contendrá el código del carácter. Asimismo, realiza una llamada al gancho situado en &HFD02 después de apilar los pares HL, DE y BC. No modifica ningún registro.

#### **Posición: &HA2**

Imprime el carácter del acumulador en la posi-



ción en la que se encuentre el cursor, aunque se trate de un código de control. Actualiza la pantalla, desplazándola o haciendo un cambio de línea si es preciso. Después de apilar todos los registros ealta al gancho situado en &HFDA4. No modifica ningún registro pero sí las coordenadas Y y X del cursor (almacenadas en &HF3DC y &HF3DD respectivamente) y la dirección &HF661.

#### **Posición: &HAB**

Envía el carácter contenido en el acumulador a la impresora, esperando hasta que ésta esté preparada. Si se pulsa CTRL-STOP el flag C se pondrá a 1. No modifica ningún registro.

#### **Posición: &HAB**

Esta rutina es llamada por la anterior. Su finalidad es comprobar si la impresora está ON-LINE. De no ser así el flag Z se pondrá a 1. Modifica el par AF.

#### **Posición: &HAB**

Transforma el código contenido en el acumulador en un carácter gráfico (si es menor que 32), en la forma que el VDP está preparado para aceptar. Prueba con VPOKE 0,1 y entenderás perfectamente el funcionamiento de esta rutina. Modifica el par AF.

#### **Posición: &HAE**

Acepta una línea completa del teclado. Puede que una línea puede contener hasta 255 caracteres, ésta se almacena en buffer de entrada que está situado entre las posiciones &HF55E y &HF66D. A la salida, el par HL apunta al inicio de este buffer menos uno. Modifica todos los registros.

#### **Posición: &HBI**

Esta rutina es similar a la anterior. Aceptará la entrada de caracteres e irá mostrándolos en la pantalla hasta que se pulse RETURN o CTRL-STOP. Modifica todos los registros.

#### **Posición: &HB4**

Esta rutina actúa de forma idéntica a las anteriores, pero visualizando antes el signo de interrogación característico de los INPUT.

#### **Posición: &HB7**

Sirve para comprobar si se ha pulsado CTRL-STOP. Si esto es así, el flag C se pondrá a 1. Modifica AF.

#### **Posición: &HBA**

Esta rutina complementa a la anterior, pero además analiza si se ha pulsado únicamente la tecla STOP, para detener la ejecución del programa cuando así sea. Altera el par AF.

#### **Posición: &HBD**

Esta rutina hace exactamente lo mismo que la anterior, pero empleando más tiempo.

#### **Posición: &HCO**

Produce un BEEP e inicializa el PSG, llamando a la rutina situada en &H90. Modifica todos los registros. Su equivalente en BASIC sería: BEEP.

#### **Posición: &HC3**

Su cometido es borrar la pantalla, con la condición de que pongas el flag Z a 0 antes de llamarla. Modifica los pares AF, BC y DE y las posiciones de la RAM del sistema relacionadas con el cursor. El modo de pantalla que se esté utilizando es indiferente.

#### **Posición: &HC6**

Sitúa el cursor en la posición especificada por el registro HL, para lo cual es necesario poner la columna en H y la fila en L. Altera el par AF y las direcciones de memoria encargadas de guardar las coordenadas de cursor (&HF3DC y &F3DD). Su equivalente en BASIC sería: LOCATE L, H.

#### **Posición: &HC9**

Esta rutina es llamada por el intérprete BASIC para saber si las teclas de función están activas.

#### **Posición: &HCC**

Se llama a esta rutina para desconectar la visualización de las teclas de función. Su equivalente en BASIC sería: KEYOFF. Altera AF, BC y DE.

#### **Posición: &HCF**

Puede utilizarse para mostrar el contenido de las teclas de función en la pantalla. Actúa como la instrucción BASIC KEYON. Modifica los registros AF, BC y DE, así como la posición &HF3DE, que será cargada con &HFF.

#### **Posición: &HD2**

Esta rutina se emplea para cambiar de pantalla y ponerla en el otro modo de texto.

#### **Posición: &HD5**

Esta llamada realiza una función idéntica a la instrucción BASIC A=STICK(A), por lo que te sugiero que leas el manual de tu ordenador para conocer los detalles. Modifica todos los registros.

#### **Posición: &HD8**

Analiza el estado del disparador especificado por un número que debe cargarse en el acumulador. A la salida, tendrás un cero en el registro A, si ha habido algún disparo, o 255, si no se ha pulsado el disparador. Modifica AF.

#### **Posición: &HDB**

Esta rutina funciona de forma análoga a la instrucción BASIC PAD (A). Por consiguiente, te aconsejo que mires allí para obtener una información completa. Altera todos los registros.

#### **Posición: &HDE**

Esta rutina lee la raqueta de juegos especificada por el registro A. Asimismo, devuelve en el acumulador un parámetro comprendido entre 0 y 255, referido a la posición actual. Modifica todos los registros.

#### **Posición: &HE1**

Con esta llamada pondrás el motor del cassette en marcha y podrá leer la cabecera. Si se pulsa CTRL-STOP el flag C se pondrá a 1. Modifica todos





los registros.

**Posición: &HE4**

Se emplea para leer un byte de la cinta, que será devuelto en el acumulador. Al igual que la rutina anterior, el carry se encenderá si la operación es abortada. Modifica todos los registros.

**Posición: &HE7**

Esta rutina sirve para detener la operación de lectura del cassette. No altera ningún registro.

**Posición: &HEA**

Esta rutina pone el motor del cassette en marcha y escribe la cabecera en la cinta. El carry se pondrá a 1 si se interrumpe la escritura. Modifica todos los registros.

**Posición: &HED**

Carga el acumulador con un dato y esta rutina te lo escribirá en la cinta. Como siempre el carry encendido te indicará si la operación fue abortada por la pulsación de CTRL-STOP. Modifica todos los registros.

**Posición: &HF3**

Esta rutina conectará el motor del cassette, si el acumulador contiene un 1, o lo parará, si contiene un 0. Por otra parte, si cargase el registro A con &HFF, antes de llamarla, invertirá el estado del motor.

**Posición: &HFC**

Esta rutina desplaza al cursor gráfico un punto hacia la derecha. Al llamarla, la posición &HF92A y siguiente debe contener la dirección de la VRAM en la que se encuentra el punto. Asimismo, deberá po-

ner la en posición &HF92C un valor cuyo único bit encendido muestre el punto a tratar. Por consiguiente si &HF92C contiene un 32 (&B00001000) el cursor gráfico señalará al tercer punto de la posición especificada por &HF92A, al volver de la rutina. Modifica el par AF y las tres posiciones de memoria antes referidas.

**Posición: V &HFF**

Esta rutina hace exactamente lo mismo que la anterior, sólo que el cursor gráfico se desplaza un punto a la izquierda.

**Posición: &H102**

Hace lo mismo que las anteriores pero desplazando el cursor hacia arriba.

**Posición: &H105**

Trabaja igual que la rutina anterior pero pone el carry a 1 si se alcanza la fila superior de la pantalla.

**Posición: &H108**

Se comporta como &HFC pero bajando un punto el cursor gráfico.

**Posición: &H10B**

También hace bajar un punto el cursor gráfico, aunque pondrá el carry a 1 si se llega a la fila inferior de la pantalla. El resto como &HFC.

**Posición: &H11D**

Esta rutina devuelve en el acumulador el código de color del punto señalado por las posiciones de memoria &HF92A a &HF92C (ver la rutina situada en &HFC).

**Posición: &H123**

Esta rutina traza una línea hacia la derecha a partir de la posición especificada por las direcciones &HF92A a &HF92C (ver la rutina situada en &HFC) y la longitud contenida en HL. El color del trazo ha de colocarse en &HF3F2. Modifica todos los registros.

**Posición: &H132**

Usando esta rutina actuarás directamente sobre el diodo de las mayúsculas. Así, si el acumulador contiene un cero lo encenderá, con otro valor, lo apagarás. Modifica el par AF.

**Posición: &H141**

Esta rutina comprueba el estado de la matriz del teclado. Dicha matriz forma un cuadrado de 8x8. El acumulador deberá contener el número de la fila a explotar. A la salida tendrá que A tiene un 255, si no ha sido pulsada ninguna tecla de la fila en cuestión, o un bit puesto a cero, indicando la tecla que sí se ha pulsado. Únicamente altera el par AF y no espera hasta que se pulsa una tecla.

**Posición: &156**

Sirve para borrar completamente el buffer del teclado. Modifica el registro HL.

Nota: Las posiciones de la ROM 8 y 7 contienen los números de los puertos asignados para las operaciones de entrada/salida al VDP.



X(W)?A AND X  
THEN 960 AND  
420 NEXT W: RETURN

## VARIABLES ROM DEL SISTEMA

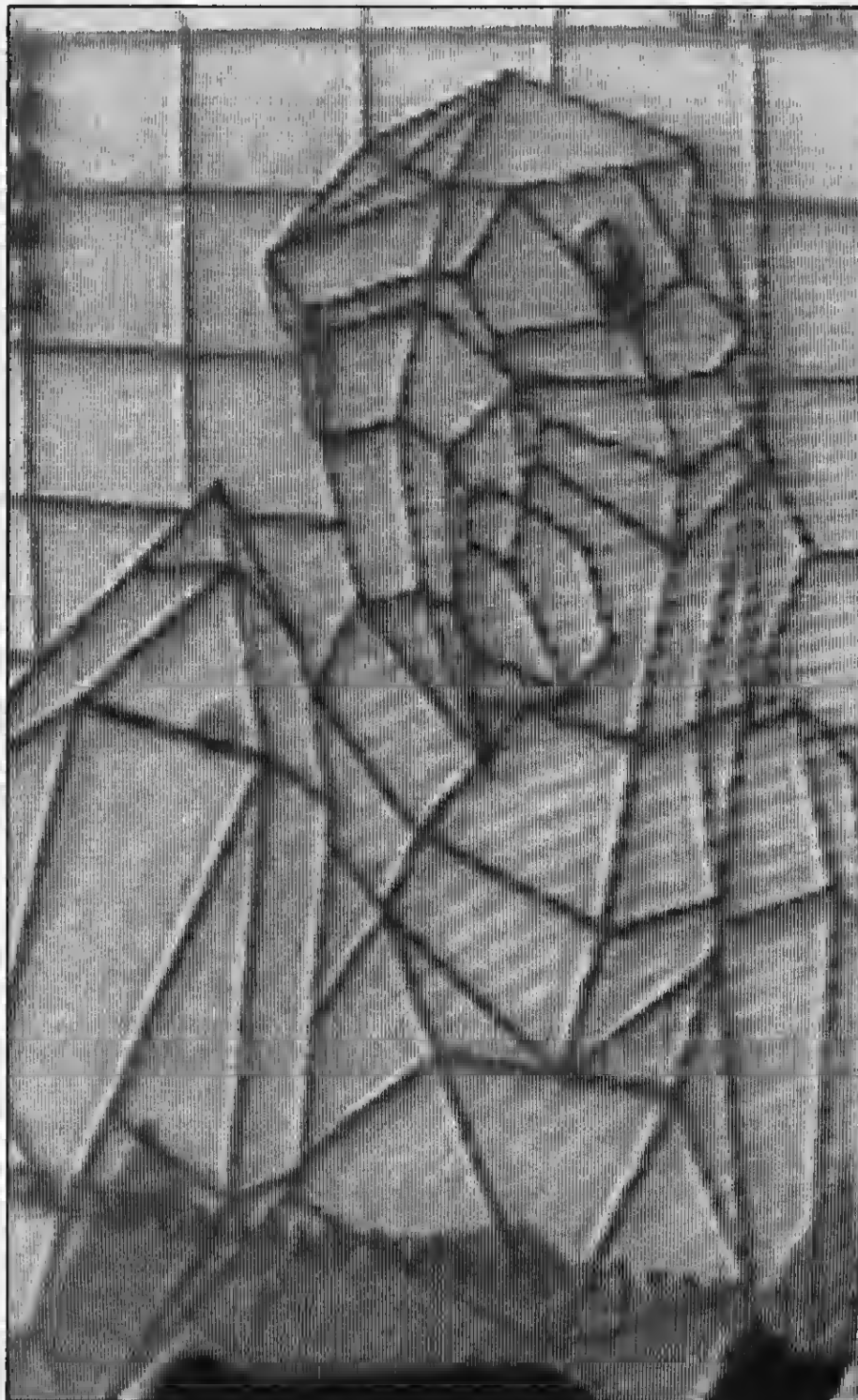
### DIRECCION

#### FUNCION

- 003E Inicializar teclas funcionales.  
MODIFICA Todos los registros.
- 004A Leer datos de la VRAM  
ENTRADA HL: dirección VRAM  
SALIDA A: datos  
MODIFICA AF
- 004D Escribir datos en la VRAM  
ENTRADA HL: dirección VRAM  
A: datos  
MODIFICA AF
- 0056 Introducir una constante en la VRAM  
ENTRADA BC: longitud  
HL: dirección VRAM  
A: datos  
MODIFICA AF, BC
- 0059 Transferir un bloque de la memoria principal a la VRAM  
ENTRADA BC: longitud  
DE: dirección RAM de destino  
HL: dirección VRAM de origen  
MODIFICA Todos los datos
- 005C Transferir un bloque de la memoria principal a la VRAM  
ENTRADA BC: longitud  
DE: dirección VRAM de destino  
HL: dirección RAM de origen  
MODIFICA Todos los registros
- 0090 Inicializar el generador programable de sonidos (PSG)  
MODIFICA Todos los registros
- 0093 Escribir datos en el PSG  
ENTRADA A: n.º del registro
- 0098 Leer datos del PSG  
ENTRADA A: n.º de registro  
SALIDA A: datos  
MODIFICA A
- 009C Verificar buffer de teclado  
SALIDA Cero (flag) si el buffer está vacío
- 009F Esperar una entrada de teclado  
SALIDA A: el carácter  
MODIFICA AF
- 00D6 Examinar estado del joystick  
ENTRADA A: stick ID (0-2)  
SALIDA A: stick status (0-8)  
MODIFICA Todos los registros
- 00D8 Examinar disparador  
ENTRADA A: disparador ID (0-4)  
SALIDA A: 255 si está pulsado  
MODIFICA AF
- 0141 Obtener el estado de la matriz del

teclado  
ENTRADA A: dirección de la fila  
SALIDA A: estado de la fila

MODIFICA AF  
0158 Borrar buffer de teclado  
MODIFICA HL





# H. VARIABLES RAM DEL SISTEMA

## DIRECCION

### FUNCION

F360 rutina para leer la ranura primaria  
 F365 rutina para escribir en la ranura primaria  
 F36C llamar rutina de la ranura primaria  
 F39A dirección inicial para USRO-9  
 F3AE longitud de línea = 39  
 F3AF longitud de línea = 31  
 F3B0 longitud de línea  
 F3B1 líneas en pantalla = 24  
 F3B2 espacio de columna = 14  
 F3B3 SCREEN 0 tabla de nombres  
 F3B5 tabla de colores  
 F3B7 forma de carácter  
 F3B9 atributo  
 F3BB sprite  
 F3BD SCREEN 1 tabla de nombres  
 F3BF tabla de colores  
 F3C1 forma de carácter  
 F3C3 atributo  
 F3C5 sprite  
 F3C7 SCREEN 2 tabla de nombres  
 F3C9 tabla de colores  
 F3CB forma de carácter  
 F3CD atributo  
 F3CF sprite  
 F3D1 SCREEN 3 Tabla de nombres  
 F3D3 tabla de colores  
 F3D6 forma de carácter  
 F3D7 atributo  
 F3D9 sprite  
 F3DB enganche de tecla  
 F3DC coord. Y cursor  
 F3DD coord. X cursor  
 F3DE teclas funcionales  
 F3DF contenido del registro VDP  
 F3E7 = 0  
 F3E6 = (FF)  
 F3E9 color de primer plano  
 F3EA color de fondo  
 F3EB color de borde  
 F3EC salto 0  
 F3EF salto 0  
 F3F2 byte atributo  
 F3F3 dirección de tabla de espera  
 F3F6 = (FF)  
 F3F6 sincronización de exploración de teclas  
 F3F7 = 50  
 F3F6 (put) buffer teclado  
 F3FA (get) buffer teclado  
 F3FC parámetros de E/S cassette  
 F40F puntero de RESUME TEXT  
 F414 código de error  
 F415 cabeza impresora  
 F416 salida impresora  
 F417 0 = para impresora MSX  
 F418 distinto de cero para salida de caracteres sin procesar

F419 función val  
 F41C línea cursor  
 F41F buffer de proceso  
 F55D coma para INPUT  
 F55E buffer de entrada de teclado  
 F660 fin de buffer  
 F661 posición terminal  
 F662 flag de matriz  
 F663 tipo de valor  
 F664 tipo de operador  
 F666 para proceso  
 F666 puntero de texto para getch  
 F666 forma interna de la constante posterior a getch  
 F669 tipo de constante  
 F672 parte superior de la memoria  
 F674 parte superior de la pila  
 F676 parte superior del texto  
 F676 descripción temporal  
 F67A almacenar descripciones temporales  
 F696 descripción de cadena después de operaciones  
 F69B parte superior posible del espacio de cadenas  
 F66D para operaciones de reorganización de datos  
 F6A1 puntero de sentencia FOR  
 F6A3 puntero de sentencia DATA  
 F6A5 flag para FOR Y USR  
 F6A6 flag para INPUT Y READ  
 F6A7 para sentencias  
 F6A9 = 0 cuando no hay línea de programa  
 F6AA = 0 en modo AUTO  
 F6AD incremento en AUTO  
 F6AF puntero de texto para RESUME  
 F6B1 grabar pila para proceso de errores  
 F6B3 línea de error  
 F6B6 línea de curso  
 F6B7 puntero de texto para RESUME  
 F6B9 línea de proceso de errores  
 F6BB = 1 si se está procesando un error  
 F6BC tareas temporales  
 F6B6 antiguo n.º de línea establecido por CRTLSTOP, STOP Y END  
 F6C0 antiguo puntero de texto  
 F6C2 dirección inicial de variables simples  
 F6C4 dirección inicial de matrices  
 F6C6 fin de la memoria utilizada  
 F6C6 puntero DATA  
 F6CA tipo de variable para A-Z  
 F6E4 pila usada en labores de recogida de basura  
 F6E6 longitud de tabla  
 F6E6 tablas de parámetros para funciones definidas para el usuario  
 F74C puntero de bloqueo de parámetros  
 F74E longitud del bloqueo de parámetros  
 F750 direcciones de los parámetros

F7B4 flag para búsqueda de parámetros  
 F7B5 fin de búsqueda  
 F7B7 = 0 si no corresponde función  
 F7BA uso temporal en recogida de basura  
 F7BC para uso de intercambios  
 F7C4 = 0 para rastreo desactivado  
 F7C5 = zona de trabajo para rutinas de paquetes BCD  
 F83F = zona de datos para manipulación de ficheros  
 F87F contenido de teclas funcionales  
 F91F tablas de VRAM BASE  
 F92A para GENGRP  
 F931 zona de trabajo y CIRCLE  
 F949 zona de trabajo de PAINT  
 F956 zona de trabajo de PLAY  
 FBBO posible recalentamiento si es distinto de cero  
 FBB1 distinto de cero si el texto BASIC está en ROM  
 FBB2 tabla de terminadores de línea  
 FBCA primera posición de carácter en INLIN  
 FBCC código para cursor  
 FBCE flag para teclas funcionales  
 FBCE flags para interruptores condicionales por teclas de función  
 FBD6 flag de condición  
 FBD9 flag de enganche  
 FBDA antiguo estado de tecla  
 FBE5 nuevo estado de tecla  
 FBFO buffer de código de tecla  
 FC16 operaciones de proceso de pantalla  
 FC40 operación de pattern converter  
 FC46 parte inferior de la RAM  
 FC4A parte superior de la memoria  
 FC4C tabla de interrupción  
 FCA9 RTYCNT  
 FC9B INTFLG  
 FC9C PAD X  
 FC9D PAD Y  
 FC9E JIFFY  
 FCA0 intervalo  
 FCA2 contador de intervalo  
 FCA4 leer cassette  
 FCA6 encabezamiento de carácter gráfico  
 FCA7 contador de secuencia de escape  
 FCA8 flag de inserción  
 FCA9 ON/OFF cursor  
 FCAA carácter de cursor  
 FCAB estado de la tecla CAP6  
 FCAC operaciones de la tecla desactivada  
 FCAD no utilizada  
 FCAE = 0 mientras se carga un programa BASIC  
 FCAF modo de pantalla (screen)  
 FCB0 antiguo modo screen  
 FCB1 carácter para CAS:  
 FCB2 color de borde en PAINT  
 FCB3 cursor gráfico, coord. X  
 FCB5 cursor gráfico, coord. Y  
 FCB7 acumulador gráfico, X  
 FCB9 acumulador gráfico, Y  
 FCB6 flag de DRAW  
 FCBC escala en DRAW  
 FCBD ángulo de DRAW  
 FCBE BLOAD/BSAVE  
 FCBF inicio de BSAVE  
 FCCI zona de trabajo de ranura  
 FD9A enganches



# CODIGO MAQUINA, IMPRESO Y PRET A PORTER

No hay mucha literatura escrita para el MSX sobre el código máquina (cinco libros en el momento de cerrar esta edición). Sin embargo, dado que existen muchos ordenadores en el mercado que usan el Z80 como microprocesador, no resulta difícil encontrar libros que faciliten información sobre este lenguaje. De cualquier forma, confío en que esta pequeña reseña os sea útil, a la hora de comprar un manual:

**Título:** MSX código máquina. Programación práctica.

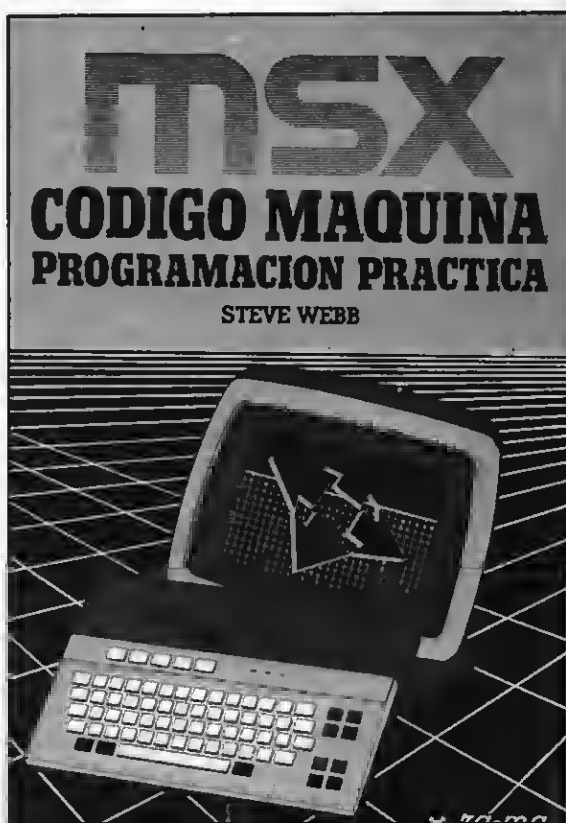
**Autor:** Steve Webb.

**Editorial:** RAMA

**Páginas:** 128

**Precio aproximado:** 1.200 ptas.

Se trata de un libro pequeño, en el que no hay sitio para explicar con demasiada profundidad el funcionamiento del Z80. En realidad la mayor parte de las páginas están dedicadas al procesador de vídeo. No me gustó.



Joe Pritchard

## Lenguaje Máquina para MSX

Introducción y  
Conceptos avanzados



**Título:** Lenguaje máquina para MSX.

**Autor:** Joe Pritchard

**Editorial:** ANAYA MULTIMEDIA

**Páginas:** 240

**Precio aproximado:** 1.500 ptas.

Si eres neófito en el C.M. este libro te interesará, puesto que explica con cierto detalle los diferentes nemónicos de Z80, además del funcionamiento del VDP y del PSG. No está mal.

**Título:** MSX. Lenguaje máquina

**Autores:** Dullin & Straeeenburg

**Editorial:** DATA BECKER - Ferrer Moret

**Páginas:** 312

**Precio aproximado:** 2.200 ptas.

Al igual que el anterior, este libro es aconsejable para los principiantes en el C.M. Incluye una relación detallada de los diferentes nemó-

# MSX

## Lenguaje Máquina

**UN LIBRO DATA BECKER**

EDITADO POR FERRE MORET, S.A.

nicos y da buenos consejos. Asimismo, contiene algunos programas útiles, como un deensamblador y un simulador, escritos casi totalmente en BASIC (jufi). Es un buen libro.

**Título:** Guía del programador MSX

**Autores:** Burkinshaw & Goodley

**Editorial:** RAMA

**Páginas:** 208

**Precio aproximado:** 1.800 ptas.

El título no engaña, puesto que se trata de una verdadera guía del programador. Incluye una descripción detallada del BASIC MSX, del VDP, del PSG, de la arquitectura del ordenador y del funcionamiento del microprocesador, además de un excelente programa de utilidad para generar SPRITES. Hay que decir que todo esto se consigue gracias a una letra inusualmente pequeña. Es un libro imprescindible, tanto para el experto como para el principiante.



**GUÍA DEL PROGRAMADOR MSX**

# MSX

## Guía del programador y manual de referencia



**Título:** MSX. Guía del programador y manual de referencia

**Autores:** Sato, Mapstone & Muriel

**Editorial:** ANAYA MULTIMEDIA

**Páginas:** 702 (!)

**Precio aproximado:** 2.250 ptas.

Disponiendo de tal cantidad de páginas se podrían tratar todos los temas. Sin embargo, no hace mención a los nemónicos del Z80, aunque describe exhaustivamente las interioridades del BASIC. Esto resultará ser un inconveniente para los principiantes que deseen aprender a programar en C.M. No obstante, resultará ideal para los expertos en otros ordenadores que quieran adentrarse rápidamente en la arquitectura del sistema MSX. Es, pues, un libro para iniciados. La información sobre el BIOS, los ganchos y la RAM del sistema, que se da en las últimas cien páginas, sólo puede encontrarse en los manuales, casi secretos, de los distintos fabricantes.

No puede concluir sin caer en la tentación de mencionar un libro escrito para el ZX81 y para el SPECTRUM, por Joan Sales Roig (no, no es amigo mío), titulado precisamente «Programación en código máquina para el ZX-81 y para el Spectrum». Quizá os estéis preguntando qué tienen que ver estos ordenadores con el MSX. Pues bien, simplemente todos emplean el Z80 como microprocesador, lo que hace que las diferencias entre ellos, a nivel de C.M., sean pequeñas. Creed que es el mejor libro para principiantes en el código máquina que he podido ver (y he tenido la suerte de ver bastantes). Gracias a él ahorraréis horas de aprendizaje y la mayoría de los «¿qué pasaría si...?» serán contestados. Es de la editorial REDE y cuesta unas 1.400 ptas.





# DESEMSAMBLADOR

```

1000 *****
1010 *
1020 * DESEMSAMBLADDR *
1030 *
1040 * POR J. VICEIRA *
1050 * PARA MSX-EXTRA *
1060 *****
1070 SCREEN 0,1:KEY OFF:WIDTH 35:COLOR 1,5,5:CLS
1080 ON STDP GOSUB 3690:STDP DN
1090 GOSUB 3360
1100 GDSUB 3530
1110 CLS
1120 FP=0:FV=0:DI$="0":OF$=""
1130 LDCATE 5,10:PRINT "¿Desea copia impresa?";
1140 A$=INPUT$(1)
1150 IF A$<>"S" AND A$<>"s" AND A$<>"N" AND A$<>"n" THEN 1140
1160 IF A$="S" OR A$="s" THEN FP=1
1170 CLS:LDCATE 5,10
1180 INPUT "¿Direccion de comienzo";DI$
1190 H$=RIGHT$(DI$,1)
1200 IF H$="H" OR H$="h" THEN D=VAL("&h"+LEFT$(DI$,LEN(DI$)-1)) ELSE D=VAL(DI$)
1210 LDCATE 5,14:INPUT "¿Direccion final";OF$
1220 IF OF$="" THEN FV=1:OF$=HFFFF;GOTO 1250
1230 H$=RIGHT$(OF$,1)
1240 IF H$="H" OR H$="h" THEN DF=VAL("&h"+LEFT$(OF$,LEN(OF$)-1)) ELSE DF=VAL(OF$)
1250 CLS
1260 A$="Direc. C. Assembler C. M aquina"
1270 PRINT A$:PRINT STRING$(33,42)
1280 IF FP THEN LPRINT A$:LPRINT STRING$(33,42)
1290 "
1300 D$=HEX$(D)
1310 IF LEN(D$)<4 THEN D$="0"+D$:GOTO 1310
1320 P=PEEK(D)
1330 P$=HEX$(P)
1340 PB$=BIN$(P)
1350 GOSUB 1710
1360 S=1
1370 B2$=HEX$(PEEK(D+1))
1380 IF LEN(B2$)<2 THEN B2$="0"+B2$
1390 B3$=HEX$(PEEK(D+2))
1400 IF LEN(B3$)<2 THEN B3$="0"+B3$
1410 B4$=HEX$(PEEK(D+3))
1420 IF LEN(B4$)<2 THEN B4$="0"+B4$
1430 IF P$="C8" OR P$="DD" OR P$="E0" OR P$="FD" THEN GOSUB 1790 ELSE GDSUB 1740
1440 L$=D$+STRING$(2,32)+I$+STRING$(16-LEN(I$),32)
1450 FOR J=1 TO S
1460 A$=HEX$(PEEK(D+J-1))
1470 IF LEN(A$)<2 THEN A$="0"+A$
1480 L$=L$+A$+CHR$(32)
1490 NEXT J
1500 PRINT L$
1510 IF FV THEN A$=INPUT$(1):GOTO 1

```

```

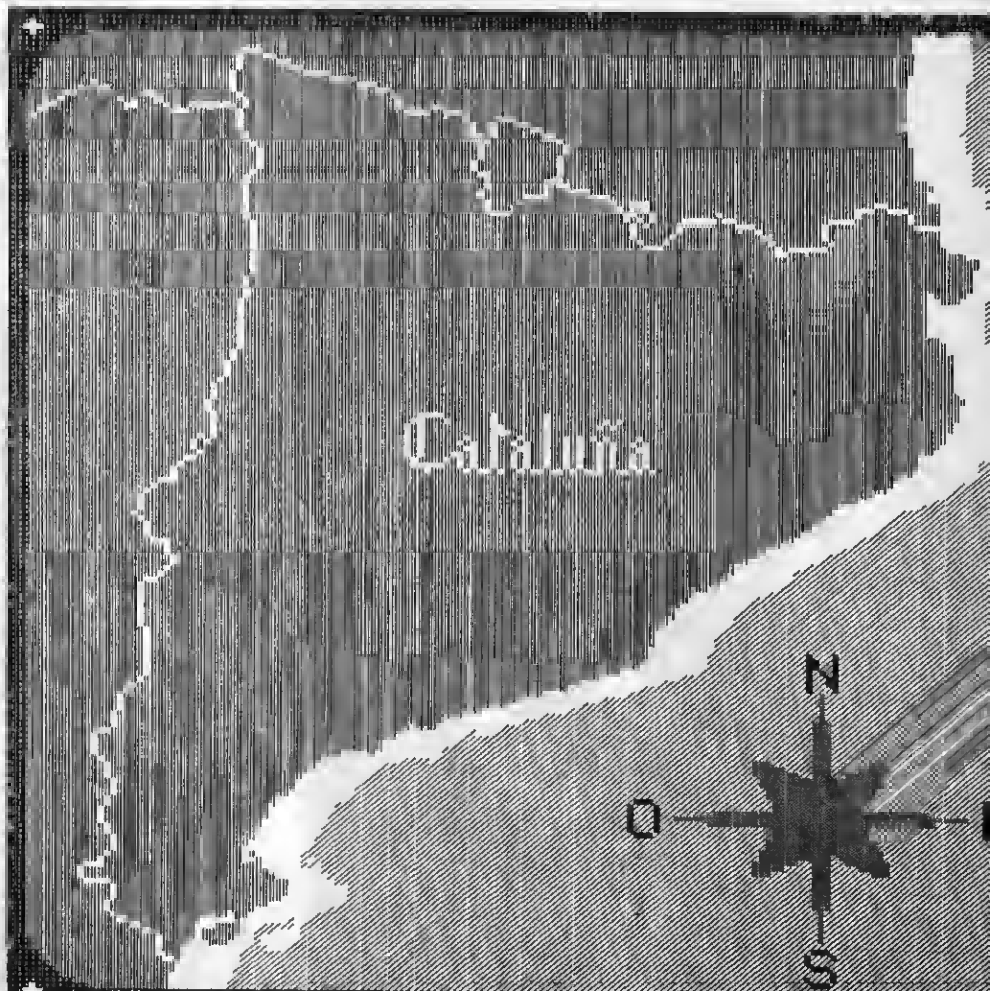
550
1520 D=D+S
1530 IF D<=DF THEN 1300 ELSE 1560
1540 "
1550 IF A$<>"F" AND A$<>"f" THEN D=D+S:GOTO 1300
1560 PRINT
1570 PRINT "Pulse una tecla para volver a empezar o 'I' para instru-

```

```

cciones.";
1580 A$=INPUT$(1)
1590 IF A$="I" OR A$="i" THEN CLS:GOSUB 3620
1600 GOTO 1110
1610 "
1620 A=VAL("&b"+LEFT$(PB$,2))
1630 B=VAL("&b"+MID$(PB$,3,3))
1640 C=VAL("&b"+RIGHT$(PB$,3))

```





(W)+10 AND 410 IF Y(W)  
Y(W) <B+30 AND +5>B

```
1650 RETURN
1660 '
1670 I$=I$+"("+"M$"+"+"+83$+"")":RETURN
1680 '
1690 I$=I$+"("+"M$"+"+"+B4$+"")":RETURN
1700 '
1710 IF LEN(P8$)<8 THEN P8$="0"+P8$
:GOTO 1710
1720 RETURN
1730 'CODIGOS DE OPERACION DE 1 BYT
E
1740 GOSUB 1620
1750 ON A+1 GOSUB 1850,2190,2220,22
30
1760 RETURN
1770 JR 2,32803!
```

```
1780 'CODIGOS DE OPERACION DE 2 BYT
ES
1790 S=2
1800 IF P$="C8" THEN GOSUB 2540
1810 IF P$="DD" THEN M$="IX":GOSUB
2630
1820 IF P$="FO" THEN M$="IY":GOSUB
2630
1830 IF P$="EO" THEN GOSUB 3040
1840 RETURN
1850 ON C+1 GOSUB 1870,1940,1960,20
40,2060,2070,2080,2090
1860 RETURN
1870 ON 8+1 GOSUB 1890,1900,1910,19
20,1930,1930,1930,1930
1880 RETURN
1890 I$="NOP":RETURN
1900 I$="EX AF,AF":RETURN
1910 I$="OJNZ "+B2$:S=2:RETURN
1920 I$="JR "+B2$:S=2:RETURN
1930 I$="JR "+C$(8-4)+", "+B2$:S=2:R
ETURN
1940 IF 8/2=INT(8/2) THEN I$="LD "+
RO$(B/2)+", "+B3$+B2$:S=3 ELSE I$="A
DD HL, "+RD$(INT(8/2))
1950 RETURN
1960 ON B+1 GOSUB 1980,1980,1980,19
80,2000,2010,2020,2030
1970 RETURN
1980 IF 8/2=INT(8/2) THEN I$="LO ("
+RO$(B/2)+", "+A" ELSE I$="LD A, ("+"RD
$(INT(8/2))+", "+A"
1990 RETURN
2000 I$="LO ("+"B3$+B2$+", "+HL":S=3:R
ETURN
2010 I$="LO HL, ("+"B3$+B2$+", "+S=3:R
ETURN
2020 I$="LO ("+"B3$+B2$+", "+A":S=3:RE
TURN
2030 I$="LD A, ("+"B3$+B2$+", "+S=3:RE
TURN
2040 IF 8/2=INT(8/2) THEN I$="INC "
+RD$(B/2) ELSE I$="DEC "+RO$(INT(B/
2))
2050 RETURN
2060 I$="INC "+R$(B):RETURN
2070 I$="OEC "+R$(B):RETURN
2080 I$="LD "+R$(8)+", "+B2$:S=2:RET
URN
2090 ON 8+1 GOSUB 2110,2120,2130,21
40,2150,2160,2170,2180
2100 RETURN
2110 I$="RLCA":RETURN
2120 I$="RRCA":RETURN
2130 I$="RLA":RETURN
2140 I$="RRA":RETURN
2150 I$="DAA":RETURN
2160 I$="CPL":RETURN
2170 I$="SCF":RETURN
2180 I$="CCF":RETURN
2190 I$="LD "+R$(B)+", "+R$(C)
2200 IF 8=6 AND C=6 THEN I$="HALT"
2210 RETURN
2220 I$=AL$(B)+R$(C):RETURN
2230 ON C+1 GOSUB 2250,2260,2350,23
60,2450,2460,2510,2520
2240 RETURN
2250 I$="RET "+C$(8):RETURN
2260 ON B+1 GOSUB 2280,2310,2280,23
20,2280,2330,2280,2340
2270 RETURN
2280 I$="POP "+RO$(8/2)
2290 IF 8=6 THEN I$="POP AF"
2300 RETURN
2310 I$="RET":RETURN
2320 I$="EXX":RETURN
```

```
2330 I$="JP (HL)":RETURN
2340 I$="LO SP,HL":RETURN
2350 I$="JP "+C$(8)+", "+B3$+B2$:S=3
:RETURN
2360 ON B+1 GOSUB 2380,2370,2390,24
00,2410,2420,2430,2440
2370 RETURN
2380 I$="JP "+B3$+B2$:S=3:RETURN
2390 I$="OUT ("+"B2$+", "+A":S=2:RETUR
N
2400 I$="IN A, ("+"B2$+", "+S=2:RETURN
2410 I$="EX (SP),HL":RETURN
2420 I$="EX DE,HL":RETURN
2430 I$="OI":RETURN
2440 I$="EI":RETURN
2450 I$="CALL "+C$(8)+", "+B3$+B2$:S
=3:RETURN
2460 IF 8/2=INT(8/2) THEN GOSUB 248
0 ELSE I$="CALL "+B3$+B2$:S=3
2470 RETURN
2480 I$="PUSH "+RO$(B/2)
2490 IF 8=6 THEN I$="PUSH AF"
2500 RETURN
2510 I$=AL$(B)+B2$:S=2:RETURN
2520 I$="RST "+HEX$(8*8):RETURN
2530 '
2540 PB$=8IN$(VAL("&h"+B2$))
2550 GOSUB 1710
2560 GOSUB 1620
2570 ON A+1 GOSUB 2590,2600,2610,26
20
2580 RETURN
2590 I$=RT$(B)+CHR$(32)+R$(C):RETUR
N
2600 I$="BIT"+STR$(B)+", "+R$(C):RET
URN
2610 I$="RES"+STR$(B)+", "+R$(C):RET
URN
2620 I$="SET"+STR$(B)+", "+R$(C):RET
URN
2630 IF 82$="C8" THEN 2950
2640 PB$=8IN$(VAL("&h"+B2$))
2650 GOSUB 1710
2660 GOSUB 1620
2670 ON A+1 GOSUB 2690,2840,2860,28
70
2680 RETURN
2690 ON C GOSUB 2710,2750,2780,2810
,2820,2830
2700 RETURN
2710 I$="ADD "+M$+", "+RO$(INT(8/2))
2720 IF 8=5 THEN MIO$(I$,B,2)=M$
2730 IF 8=4 THEN I$="LD "+M$+", "+B4
$+B3$:S=4
2740 RETURN
2750 I$="LD ("+"B4$+B3$+", "+M$
2760 IF 8=5 THEN I$="LO "+M$+", ("+"B
4$+B3$+", "+A"
2770 S=4:RETURN
2780 I$="INC "+M$
2790 IF 8=5 THEN I$="DEC "+M$
2800 RETURN
2810 I$="INC ":GOSUB 1670:S=3:RETUR
N
2820 I$="OEC ":GOSUB 1670:S=3:RETUR
N
2830 I$="LD ":GOSUB 1670:I$=I$+", "+
B4$:S=4:RETURN
2840 IF C=6 THEN I$="LD "+R$(B)+", "
:GOSUB 1670 ELSE I$="LO ":GOSUB 167
0:I$=I$+", "+R$(C)
2850 S=3:RETURN
2860 I$=AL$(B):GOSUB 1670:S=3:RETUR
N
2870 ON C GOSUB 2890,2880,2930,2880
,2940
```

CSSJB





```

2880 RETURN
2890 I$="POP "+M$
2900 IF B=5 THEN I$="JP ("M$+)"
2910 IF B=7 THEN I$="LD SP,"+M$
2920 RETURN
2930 I$="EX (SP),"M$:RETURN
2940 I$="PUSH "+M$:RETURN
2950 PB$=BIN$(VAL("&h"+B3$))
2960 GOSUB 1710
2970 GOSUB 2220
2980 ON A+1 GOSUB 3000,3010,3020,30
30
2990 S=4:RETURN
3000 I$=RT$(B)+CHR$(32):GOSUB 1690:
RETURN
3010 I$="BIT"+STR$(B)+",";GOSUB 169
0:RETURN
3020 I$="RES"+STR$(B)+",";GOSUB 169
0:RETURN
3030 I$="SET"+STR$(B)+",";GOSUB 169
0:RETURN
3040 PB$=BIN$(VAL("&h"+B2$))
3050 GOSUB 1710
3060 GOSUB 1620
3070 ON A GOSUB 3090,3280
3080 RETURN
3090 ON C+1 GOSUB 3110,3120,3130,31
50,3170,3180,3200,3220
3100 RETURN
3110 I$="IN "+R$(B)+"(C)";RETURN
3120 I$="OUT (C),"R$(B):RETURN
3130 IF B/2=INT(B/2) THEN I$="SBC H
L,"+RD$(B/2) ELSE I$="ADC HL,"+RD$(
INT(B/2))
3140 RETURN
3150 IF B/2=INT(B/2) THEN I$="LD ("
+B4$+B3$+"),"RD$(B/2) ELSE I$="LD
"+RD$(INT(B/2))+","(B4$+B3$+)"
3160 S=4:RETURN
3170 I$="NEG":RETURN
3180 IF B THEN I$="RETI" ELSE I$="R
ETN"
3190 RETURN
3200 IF B THEN B=B-1
3210 I$="IM"+STR$(B):RETURN
3220 ON B+1 GOSUB 3240,3230,3250,32
30,3260,3270
3230 RETURN
3240 I$="LD I,A":RETURN
3250 I$="LD A,I":RETURN
3260 I$="RRD":RETURN
3270 I$="RLD":RETURN
3280 ON C+1 GOSUB 3300,3310,3320,33
30
3290 RETURN
3300 I$="LD"+N$(B-4):RETURN
3310 I$="CP"+N$(B-4):RETURN
3320 I$="IN"+N$(B-4):RETURN
3330 I$="DUT"+N$(B-4)
3340 IF B>5 THEN I$="OT"+N$(B-4)
3350 RETURN
3360 "Variables
3370 FOR J=0 TO 7
3380 READ R$(J):READ C$(J)
3390 READ RT$(J):READ AL$(J)
3400 IF J>3 THEN 3420
3410 READ RD$(J):READ N$(J)
3420 NEXT J
3430 RETURN
3440 DATA B,NZ,RLC,"ADD A","BC,I
3450 DATA C,Z,RRC,"ADC A","DE,D
3460 DATA D,NC,RL,"SUB ","HL,IR
3470 DATA E,C,RR,"SBC A","SP,DR
3480 DATA H,PO,SLA,"AND "
3490 DATA L,FE,SRA,"XOR "
3500 DATA (HL),P,SRL,"DR "
3510 DATA A,M,SRL,"CP "
3520 '
3530 PRINT TAB(6);"*****"
3540 PRINT TAB(6);"*
*"
3550 PRINT TAB(6);"* "DESENSAMBLAD
DR "*"
3560 PRINT TAB(6);"*
*"
3570 PRINT TAB(6);"* por J. VICEI
RA "*"
3580 PRINT TAB(6);"*
*"
3590 PRINT TAB(6);"*****"
3600 PRINT:PRINT "AVISO: Todos los n
umeros que aparecen están en nume
racion hexadecimal"
3610 '
3620 PRINT
3630 PRINT TAB(11);"INSTRUCCIONES"
3640 PRINT
3650 PRINT "- Poner 'H' detrás de n
umeros hexa"
3660 PRINT "- Para modo paso a paso
pulse una tecla para nueva inst
ruccion, y 'F' para finalizar"
3670 PRINT:PRINT TAB(10);"PULSE UNA
TECLA":
3680 A$=INPUT$(1):RETURN
3690 COLOR 15,4,4:CLS:LIST 1000-106
0:END

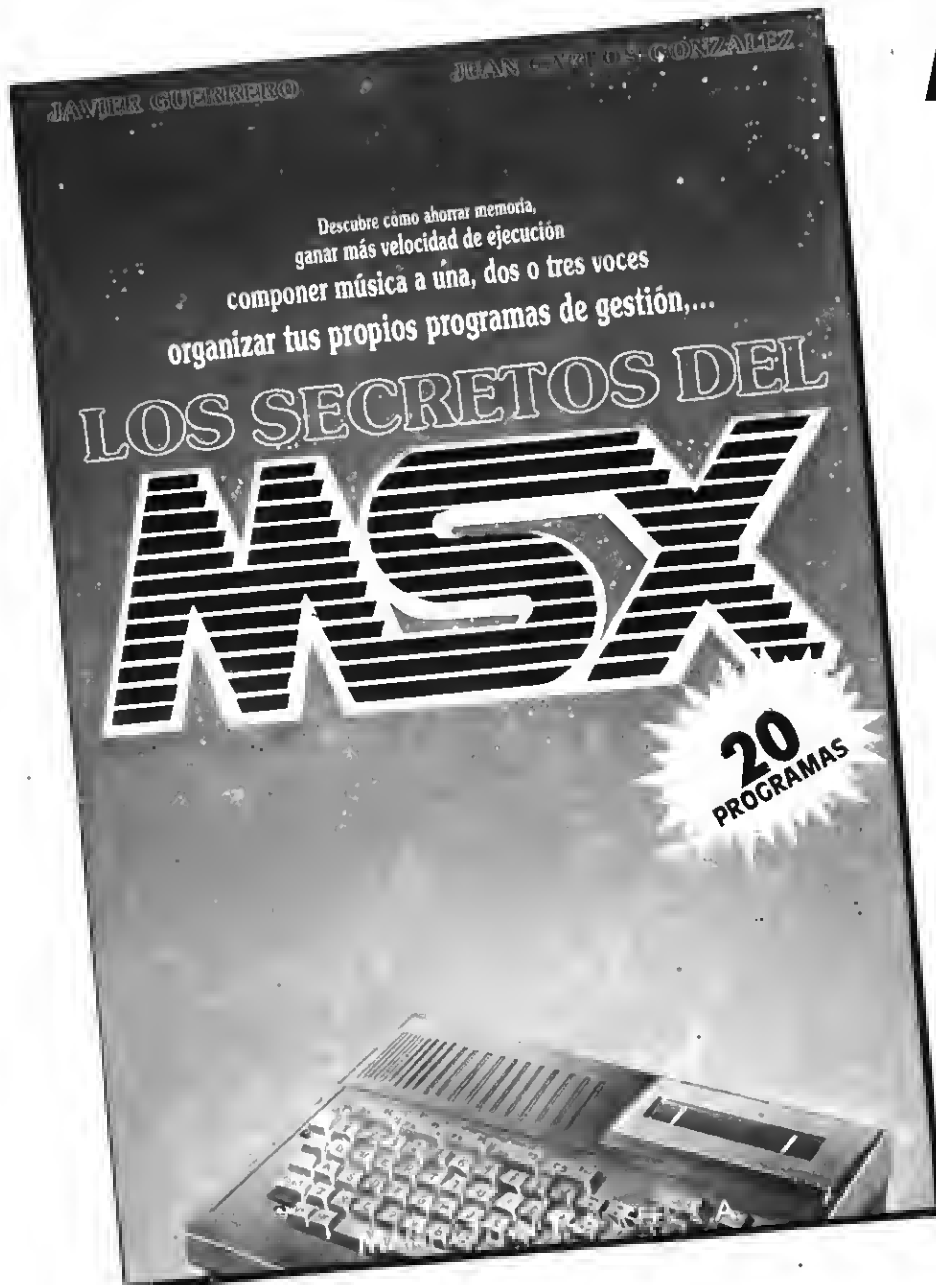
```

## Test de listado

1000 - 58	1300 -134	1600 -241	1900 -102	2200 -181	2500 -142	2800 -142	3100 -142	3400 - 40
1010 - 58	1310 -204	1610 - 58	1910 -182	2210 -142	2510 -128	2810 - 51	3110 -194	3410 -170
1020 - 58	1320 -106	1620 -151	1920 - 28	2220 - 99	2520 -139	2820 - 37	3120 - 26	3420 -205
1030 - 58	1330 -158	1630 -219	1930 -111	2230 - 42	2530 - 58	2830 -217	3130 - 85	3430 -142
1040 - 58	1340 -226	1640 -155	1940 - 82	2240 -142	2540 -209	2840 -101	3140 -142	3440 -115
1050 - 58	1350 - 79	1650 -142	1950 -142	2250 - 62	2550 - 79	2850 - 30	3150 - 24	3450 - 42
1060 - 58	1360 - 84	1660 - 58	1960 -175	2260 - 67	2560 -245	2860 - 89	3160 - 31	3460 - 48
1070 -163	1370 -160	1670 -152	1970 -142	2270 -142	2570 -254	2870 - 74	3170 - 66	3470 - 78
1080 -151	1380 -102	1680 - 58	1980 - 59	2280 -212	2580 -142	2880 -142	3180 - 47	3480 -230
1090 -168	1390 -162	1690 -153	1990 -142	2290 -195	2590 - 98	2890 -241	3190 -142	3490 - 12
1100 -114	1400 -105	1700 - 58	2000 -225	2300 -142	2600 -185	2900 -206	3200 - 30	3500 - 19
1110 -159	1410 -164	1710 - 75	2010 -225	2310 - 83	2610 -196	2910 - 15	3210 - 20	3510 - 94
1120 -213	1420 -108	1720 -142	2020 -142	2320 - 93	2620 -198	2920 -142	3220 -247	3520 - 58
1130 - 67	1430 -237	1730 - 58	2030 -142	2330 -231	2630 - 84	2930 -135	3230 -142	3530 -199
1140 - 96	1440 - 86	1740 -245	2040 -219	2340 - 91	2640 -209	2940 - 10	3240 -174	3540 -127
1150 -107	1450 -249	1750 -107	2050 -142	2350 -241	2650 - 79	2950 -210	3250 -174	3550 -131
1160 -233	1460 -169	1760 -142	2060 - 60	2360 -123	2660 -245	2960 - 79	3260 - 80	3560 -127
1170 - 12	1470 -205	1770 -102	2070 - 46	2370 -142	2670 - 73	2970 - 79	3270 - 74	3570 - 75
1180 - 26	1480 - 11	1780 - 58	2080 -107	2380 -165	2680 -142	2980 -106	3280 - 33	3580 -127
1190 - 28	1490 -205	1790 - 85	2090 - 83	2390 -187	2690 - 15	2990 - 31	3290 -142	3590 -199
1200 - 12	1500 - 1	1800 - 33	2100 -142	2400 - 10	2700 -142	3000 -220	3300 -245	3600 -188
1210 -101	1510 - 70	1810 -254	2110 -138	2410 -185	2710 - 70	3010 - 51	3310 -248	3610 - 58
1220 - 87	1520 -187	1820 - 1	2120 -144	2420 - 78	2720 -208	3020 - 62	3320 -252	3620 -145
1230 - 25	1530 - 64	1830 - 27	2130 - 71	2430 -245	2730 - 68	3030 - 64	3330 -149	3630 - 23
1240 -143	1540 - 58	1840 -142	2140 - 77	2440 -246	2740 -142	3040 -209	3340 -235	3640 -145
1250 -159	1550 -186	1850 - 22	2150 - 46	2450 -115	2750 - 45	3050 - 79	3350 -142	3650 -214
1260 -249	1560 -145	1860 -142	2160 - 71	2460 - 73	2760 - 58	3060 -245	3360 - 58	3660 -131
1270 -138	1570 -139	1870 - 47	2170 - 68	2470 -142	2770 - 31	3070 -165	3370 -189	3670 - 9
1280 -157	1580 - 96	1880 -142	2180 - 52	2480 - 37	2780 -220	3080 -142	3380 - 91	3680 - 40
1290 - 58	1590 -228	1890 - 85	2190 -134	2490 - 20	2790 -122	3090 - 44	3390 -249	3690 -223
								TOTAL:
								32639

# DESCUBRE TU ORDENADOR

## LOS SECRETOS DEL MSX



UN LIBRO PENSADO  
PARA TODOS LOS QUE  
QUIEREN INICIARSE  
DE VERDAD EN LA  
PROGRAMACION BASIC

Construcción de programas. El potente editor todo pantalla. Constantes numéricas. Series, tablas y cadenas. Grabación de programas. Gestión de archivo y grabación de datos. Tratamiento de errores. Los gráficos del MSX. Los sonidos del MSX. Las interrupciones. Introducción al lenguaje máquina.

### Y ADEMAS PROGRAMAS DE EJEMPLO

Alfabético. Canon a tres voces. Moon Germs. Bossa Nova. Blue Bossa. La Séptima de Beethoven. La Flauta Mágica de Mozart. Scrapple from the apple & Donna Lee. The entertainer. Teclee un número. Calendario perpetuo. Modificación Tabla de colores SCREEN 1. Rectángulos en 3-D. Juego de caracteres alfabéticos en todos los modos. Juego Matemático. Más grande más pequeño. Póker. Breackout. Apocalypse Now. El robot saltarin. El archivo en casa.

## EL LIBRO QUE ESPERABAS YA ESTA A LA VENTA

ENVIA HOY MISMO EL BOLETIN DE PEDIDO

Deseo me envíen el libro Los secretos del MSX, para lo cual adjunto talón de 1.500 ptas. a la orden de MANHATTAN TRANSFER, S.A.

Nombre y apellidos .....

Calle ..... n.º ..... Ciudad ..... DP .....

Este boletín me da derecho a recibir los secretos MSX en mi domicilio libre de gastos de envío o cualquier otro cargo.

**Importante:** Indicar en el sobre MANHATTAN TRANSFER, S.A.

RESERVA «LOS SECRETOS DEL MSX»

Roca i Batlle, 10-12 Bajos - 08023 BARCELONA



PARA CONECTAR  
CON EL FUTURO  
HAY QUE LLEVAR  
LA DELANTERA



LA 1.<sup>a</sup> REVISTA DE  
MSX DE ESPAÑA

